Treball de Fi de Grau

Enginyeria en Tecnologies Industrials

Millora de la gestió del rebuig d'una línia de producció

MEMÒRIA

Autor: Xavier de Pedro
Director: Lluís Solano

Convocatòria: Gener 2020

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Resum

En aquest projecte s'hi pot trobar tot l'estudi que s'ha fet sobre la línia de producció de l'empresa QIAstat-Dx®, sobre com gestionen el rebuig de línia i quines mancances té aquest rebuig i què es pot millorar. A partir d'aquest estudi preliminar es vol desenvolupar un sistema de registre per al rebuig de línia digital, des de com és el programa, fins el punt d'arribar a muntar i configurar un primer prototip.

En la part de l'estudi de la línia de producció, es tracta l'organització que té la fàbrica, és a dir, les estacions de treball que té per repartir les tasques necessàries per a la manufactura del QIAstat-Dx®. A part d'organitzar la fàbrica en estacions de treball, també es classifiquen els errors més comuns que es troben a la línia de producció i s'estandarditzen en modes de fallada per tal de poder aplicar-los en el sistema de registre digital.

A l'hora d'enfocar el problema, s'estudia quines característiques té el sistema de registre del rebuig actualment, que es fa de forma manual, amb paper i bolígraf; després d'entendre les mancances actuals, es proposen una sèrie de millores que ha de tenir un nou sistema. Aquest futur sistema plantejat és digital, i pot aportar moltes millores respecte el sistema actual.

Un cop està decidit implementar un sistema de registre digital, cal desenvolupar el projecte i explicar les eines utilitzades per aquest desenvolupament.

Primer de tot s'ha dissenyat l'estructura que vol seguir el sistema de registre, i també quin és el flux d'informació i accions que seguirà aquest. Tenint això clar s'ha decidit utilitzar el codi de programació Python i el GUI Kivy per desenvolupar l'aplicació; també es parla del llenguatge SQL, ja que el sistema utilitza base de dades per gestionar la informació mostrada i enregistrada a l'aplicació.

Després de conèixer les eines per a la programació s'ha seguit amb el desenvolupament del programa, on s'han explicat les estratègies de programació, les propietats i les funcions utilitzades més rellevants, per tal que la app tingui les funcionalitats desitjades.

Per últim s'explica com s'ha pensat un prototip molt bàsic que pugui mostrar el funcionament del sistema de registre. Per a la construcció d'aquest prototip, s'explica la tria de component i també el muntatge i la posada a punt d'aquest, que ha permès obtenir els primers resultats i impressions d'aquest sistema de registre del rebuig de línia.

Sumari

1. Glossari

Degut a la terminologia utilitzada al llarg de la memòria, el glossari permet introduir uns quants termes que s'aniran utilitzant al llarg del document. Moltes vegades aquest glossari farà referència termes anglesos, ja que a la fàbrica s'utilitza terminologia en anglès, i per tant es fa referència als elements de la fàbrica en aquest idioma.

Scrap: És el terme anglès que s'utilitza per referir-se al rebuig de la línia de producció, a QIAGEN, l'empresa on s'ha dut a terme aquest projecte es refereixen amb aquesta paraula al rebuig de línia i per això s'utilitza.

PCR: Sigles de *Polymerase Chain Reaction*, que fan referència al procés d'amplificació d'una cadena de ADN concreta per que sigui més fàcil d'identificar més endavant.

QIAstat-Dx®: Consumible de l'empresa Stat-Dx® adquirida per QIAGEN. És un cartutx de petites dimensions, que juntament amb el AM, duu a terme un procés de PCR, amb els resultats d'aquesta pot donar un diagnòstic precís i afirmar si la mostra introduïda té o no alguna de les malalties prova el QIAstat-Dx®.

AM: Sigles de *Analyzer Module*, aparell des del que es gestiona el test i que permet fer funcionar el QIAstat-Dx® per tal de dur a terme tot el procés i finalment mostrar els resultats.

Swab: Paraula anglesa per referir-se al bastonet utilitzat per agafar les mostres per al QIAstat-Dx®. És una petita peça de material amb un extrem de cotó que queda impregnat de la mostra i pot ser analitzat després.

Lisi cel·lular: Procés que trenca la membrana de les cèl·lules, deixant el material genètic d'aquestes lliures en el medi en el que es trobin.

Primers: Cadena de àcid nucleic o d'una cèl·lula relacionada que serveix com a punt de partida per a la replicació en cadena del ADN.[22]

Probes: Fragment de ADN que pot reaccionar i donar fluorescència, s'utilitza per poder identificar un resultat positiu o negatiu després de la PCR.[22]

Buffers: Paraula anglesa per referir-se a les solucions tampó, que són solucions formades per un àcid dèbil i una base conjugada, que té la característica de no veure afectat el seu PH després de l'adició d'altres àcids o bases més forts.

GTIN: Sigles de *Global Trade Item Number*, que és un número d'identificació que tenen tots els productes que estan comercialitzats a nivell global, són únics i per tant identifiquen al

producte de manera específica.

WS: Sigles de *Work Station* , que en català vol dir, estació de treball; és la manera que té la fàbrica d'identificar les màquines o grups de màquines que s'encarreguen de fer les principals tasques de manufactura del QIAstat-Dx®.

FF: Sigles de *Fluidic Film*, materia prima del cartutx.

CV: Sigles de Computer Vision, són màquines de visió artificial, que faciliten la detecció d'errors en determinats processos de manufactura.

TC: Sigles de *Transference Chamber*, una de les parts del cartutx QIAstat-Dx®.

BB: Sigles de *Bead Beater*, una de les peces del cartutx QIAstat-Dx®.

PPMixes: Barreja de *Primers* i de *Probes* que s'encarreguen de detectar el conjunt de patògens de cada producte QIAstat-Dx® diferent, es a dir, diferents panells que detecten diferents patògens.

RC: Sigles de *Reaction Chamber*, una de les parts del cartutx QIAstat-Dx®.

DCC: Sigles de *Dry Chemistry Container*, peça que conté la *master-mix* (barreja d'enzims), que és necessària per al procés que du a terme el QIAstat-Dx®.

IPC: Sigles de *In Process Control*, concepte anglès que fa referencia a un control o prova de qualitat que es fa dins del procés de manufactura.

DS: Sigles de *Discard Station* , que en català vol dir, estació de rebuig; com el seu nom indica, és el punt de la línia on es registren i es realitza el *scrap*.

MFM: Sigles de *Manufacturing Failure Mode* , que en català vol dir, modes de fallada de manufactura; serveix per codificar i ordenar tots els possibles errors que poden donar-se durant el procés de manufactura, i que ja han estat prèviament identificats i estudiats.

DataMatrix: Nom anglès per al concepte de Matriu de dades, és un sistema de codificació en 2D que permet codificar molta informació en un espai reduït.

SQL: Sigles de l'anglès *Structured Query Language*, que vol dir Llenguatge estructurat per consultes. Aquest llenguatge serveix per administrar i recuperar informació dels sistemes de base de dades.

GUI: Sigles de la paraula anglesa *Graphical User Interface*. És un sistema d'interacció entre l'ordinador i el usuari, es caracteritza per utilitzar icones i elements gràfics a part de simples textos.

2. Introducció

2.1. Origen i Motivació

Aquest projecte neix com a idea del cap de Lean de la fàbrica del QIAstat-Dx® de l'empresa QIAGEN®, que és l'encarregat de fer millores a la línia de producció per tal de millorar el rendiment d'aquesta.

Es va detectar que el registre del rebuig de línia (*scrap*) que es fa actualment a la fàbrica podia no ser del tot eficient, i per tant, es podia millorar la manera de fer el registre si es feia un registre digital d'aquest rebuig, i així apareix la idea del projecte.

El que ha motivat el projecte a tirar endavant és el fet de donar-li més importància al *scrap* dins d'una línia de producció, ja que amb la informació que s'obté a partir del producte que ha estat rebutjat es pot obtenir molta informació, com ara el funcionament de les màquines, la matèria prima o la feina dels operaris. Si s'analitza correctament pot donar molta informació i ajudar molt. A més a més el *scrap* és un fenomen que fa encarir el producte final, ja que és matèria prima desaprofitada.

Tenint present que el *scrap* aporta una informació molt important per a tot el que té a veure amb el producte, fa pensar que també és molt important la manera de que es registra i s'analitza actualment. Al ser un procés manual, triga en portar-se al departament d'enginyeria, per tant es triga en saber com està anant la producció i saber quins son els errors que estan presentant els cartutxos (que es el producte que es fabrica), i fins que el lot no s'acaba i es porten les fulles de *scrap* a les oficines, no es pot analitzar.

Al observar aquestes mancances que tenia el sistema actual de registre del rebuig, es va proposar fer un sistema que les pogués aprofitar la informació provinent del *scrap* d'una forma més eficient.

2.2. Objectius del projecte

Identificar quin problema hi ha a la fàbrica de l'empresa Stat-Dx® a l'hora de gestionar el rebuig de la línia de producció, una part inevitable de qualsevol procés productiu la qual és molt important també ja que ajuda a reduir costos del producte i fer-ho més competitiu.

Per fer-ho s'ha de començar estudiant com està organitzada la fàbrica, quines estacions de treball hi ha i quines operacions s'hi fan en cadascuna. A partir d'aquí es podran identificar els modes de fallada que hi ha en cada una de les estacions de treball, que són

els codis amb els que s'identifiquen els diferents tipus d'error que poden haver-hi a fàbrica .

Un cop estigui clar quin és el rebuig i perquè es produeix, cal estudiar què és el que s'està fent actualment a la línia per gestionar el rebuig, en aquest cas es veu que s'està fent una gestió manual feta pels operaris apuntant amb bolígraf de quina manera ha fallat el cartutx i el motiu pel qual s'ha hagut de rebutjar.

Partint d'aquest punt veurem si podem enfocar aquesta gestió del rebuig d'una manera diferent a la que es fa actualment a mà i es procurarà trobar una implementació digital que sigui més pràctica i pugui estalviar temps respecte al registre manual que es fa actualment.

Amb la implementació digital s'estima que permetrà abastir el registre de rebuig de tota la línia mitjançant més d'un dispositiu / sistema que enregistri aquest paràmetre, de forma que classifiquin les dades automàticament segons interessi (estació de treball, producte fabricat, tipus de fallida, etc.), cosa que permetria estudiar aquest rebuig de forma immediata sense haver d'analitzar les dades una a una per posar-les a la base de dades que utilitza l'empresa. Per tant també caldrà fer un estudi de quants dispositius es necessitaran realment i quines funcionalitats requerirà aquest.

El llistat de d'objectius comentats anteriorment són els següents:

- Estudiar funcionament de la línia de producció de l'empresa QIAstat-Dx® de Barcelona.
- Estudiar l'organització de la fabrica i identificar les diferents estacions de treball.
- Saber què és i com es gestiona el *scrap* de la línia de producció de Barcelona.
- Analitzar el procediment actual i definir necessitats, en conseqüència es poden proposar millores i alternatives del registre i gestió del *scrap*.
- Dissenyar un sistema que permeti complir les necessitats recollides anteriorment.
- Implementar la solució a la línia de producció per poder gestionar el *scrap* eficientment.

2.3. Abast del projecte

L'objectiu d'aquest projecte és aconseguir un sistema eficient de registre de *scrap* i que sigui viable d'aplicar-ho a la línia de producció , adequant-se a les necessitats que té la fàbrica. Aquest objectiu engloba molts camps que s'han d'analitzar per poder arribar a

tenir un sistema de registre de *scrap* que sigui eficient.

Per tant, l'abast del projecte passarà per un petit estudi de la fabricació del producte que s'està rebutjant, juntament amb una classificació i distribució de la fàbrica per tal de poder ser estudiada més endavant.

També englobarà l'estudi del rebuig de línia que hi ha actualment, com es registra, i quins errors poden trobar-se durant la fabricació.

Amb l'estudi i anàlisi previ fet, el projecte comença a desenvolupar un sistema de registre que s'adeqüi als objectius i necessitats que s'hagin triat.

Finalment es vol arribar a fer un primer prototip on el sistema de registre pugui ser implementat i provat.

3. L'empresa

La *start-up* Stat-Dx® fundada el 2010 a Barcelona, es centra en el desenvolupament de solucions diagnòstiques i també en la fabricació i distribució d'aquestes, situacions en que un diagnòstic precís i ràpid sigui imprescindible, com ara malalties infeccioses pot permetre crear una quarantena controlada en la qual es pot diagnosticar a tots els subjectes en quarantena per controlar un brot d'una d'aquestes malalties infeccioses . El sistema QIAstat-Dx® és una plataforma versàtil i fàcil d'utilitzar que consolida tècniques moleculars i d'immunoassaig en un sol dispositiu.

Després de la compra la *start-up* passa a dir-se QIAstat-Dx® l'empresa ha tingut un gran recolzament econòmic i li ha permès créixer molt i crear una oferta amb varietat de productes. L'empresa QIAstat-Dx® ha seguit amb la fàbrica de Barcelona que ha crescut molt, i també han implementat una altra fàbrica a Alemanya on també es comença a fabricar el producte.

3.1. Cartutx QIAstat-Dx® i el AM

El QIAstat-Dx® és el producte que fabrica l'empresa s'implementa el sistema de registre del *scrap*, és un cartutx que s'encarrega de realitzar un procés de PCR d'una mostra contaminada amb un patogen, i que utilitza els *primers* de diferents patògens que després d'un procés d'una hora permet donar un resultat positiu o negatius sobre els *primers* que tenia el cartutx.

Aquest consumible per tal de funcionar, ha d'anar acompanyat d'un AM, que és l'aparell encarregat d'accionar les parts mecàniques del cartutx i interaccionar amb aquest per poder realitzar el test de forma completa. Serveix per gestionar i iniciar la prova, i també té la interfície per veure els resultats de la prova del QIAstat-Dx®.

Il·lustració 1 - QIAstat-Dx® i AM

3.1.1. Funcionament del QIAstat-Dx®

Per explicar el funcionament del cartutx és necessari conèixer tots els passos que cal seguir per dur a terme un anàlisi amb un cartutx QIAstat-Dx®:

• Es comença amb una recollida de la mostra que es vulgui analitzar (per exemple, *swab*, sang, orina, etc.) aquesta s'introdueix al cartutx i es tapa.

• S'escaneja el cartutx amb un lector de codis de barres per registrar-ho al analitzador, i s'introdueix dins d'aquest, i s'executa el programa d'anàlisi.

• Elució del *swab* (si la mostra s'introdueix mitjançant un *swab*).

• Rehidratació de reactius perquè la mostra pugui fluir pel cartutx.

• Transferència de la mostra a la cambra de lisi on es produeix la Lisi cel·lular.

• Purificació d'àcids nucleics.

• Elució del lisat purificat que conté la mostra d'interès, és a dir, ADN i ARN.

• Barreja de la solució eluïda amb la *master-mix* (mescla d'enzims).

• Lliurament i dosificació a les cambres de reacció .

• Rehidratació dels *Primers* i *Probes*.

• Execució de la PCR.

• Interpretació de les dades.

3.2. Funcionament de la planta

Per poder encarar l'estudi del *scrap* cal que sàpiga quina és el funcionament de la línia de producció i com està organitzada, es a dir, quines estacions de treball hi ha i que s'hi fa a cadascuna, ja que depenent de quina feina es faci a cada estació, apareixeran diferents motius pels quals hi ha rebuig.

La planta de QIAstat-Dx® s'encarrega de fabricar un cartutx que està constituït per una desena de peces que han de ser acoblades una a una al cartutx, aquests acoblaments poden ser efectuats per robots o bé per persones, es a dir, hi ha processos manuals i automàtics, cosa que fa aparèixer l'error humà i els errors informàtics de configuració o programació. D'altra banda també mentre hi ha l'acoblament de les peces, el cartutx ha d'anar sent segellat, ja que al final ha de ser un producte estanc pel qual poden circular fluids sense que hi hagi cap tipus de filtratge. Per aquest motiu, també hi haurà passos de

la fabricació en els que s'haurà d'introduir fluids. Finalment a la línia de producció també hi ha passos en els que es fan comprovacions del estat de cartutx, ja sigui inspecció visual, com comprovació de filtratge de líquids o pesades per veure que tot el que ha de contenir el cartutx està en el producte.

3.2.1. La fàbrica

La fàbrica està situada al PCB (Parc Científic de Barcelona) al costat de les oficines, aquesta fàbrica es troba en una de les sales del PCB que està qualificada com a sala blanca, una sala que compleix una sèrie de condicions de neteja que han de seguir dins la línia de producció, tots els operaris, mesures de prevenció de contaminació, etc. També implica que per entrar a la fàbrica s'han de seguir unes normes de vestimenta estrictes, per tant, s'ha de passar per un vestuari i equipar-se adequadament.

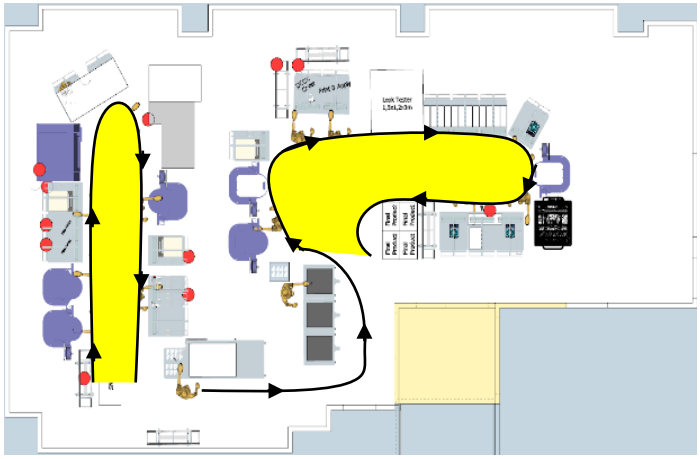
Totes aquestes condicions de sala blanca garanteixen que el producte no sigui contaminat durant el procés de fabricació, o com a mínim reduir al màxim el risc.

La fàbrica té una capacitat actual de produir 1500 cartutxos al dia, aquesta però augmenta progressivament degut al constant procés de millora que està executat a la fàbrica. Aquesta funciona en producció de lots, la mida de lot és variable segons les necessitats dels clients i la demanda que hi hagi en el moment de cada producte.

3.2.1.1. Distribució de la fàbrica, *Layout*

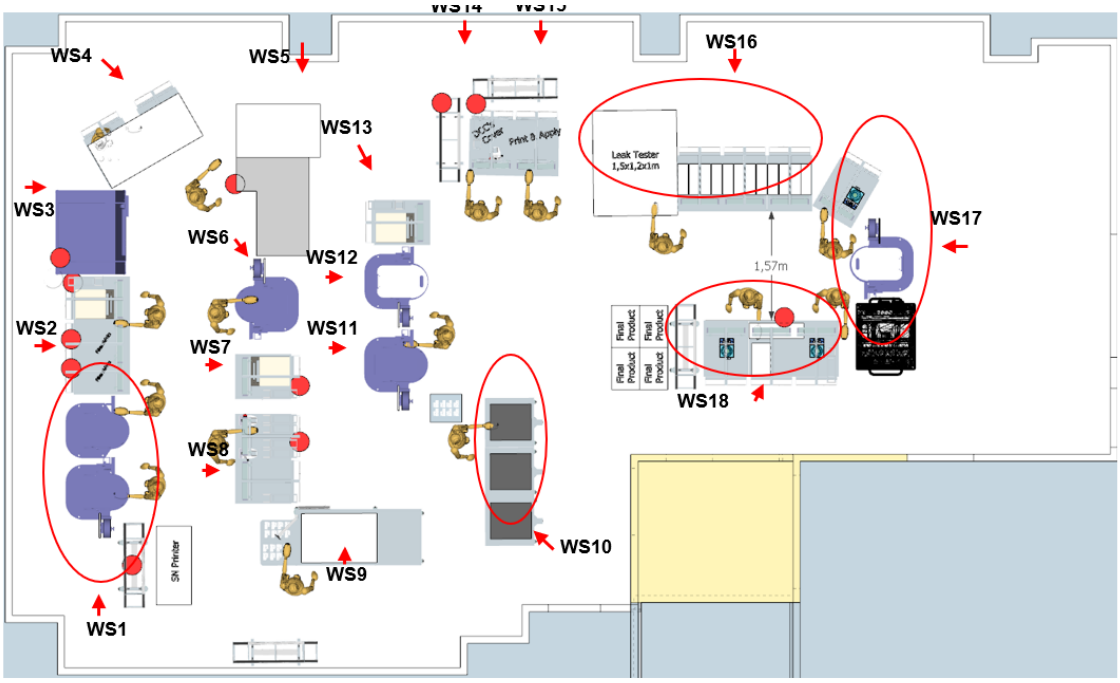
La fàbrica està distribuïda per estacions de treball (WS) en les quals a cada una s'hi realitza una tasca concreta.

La línia de producció, ha estat dissenyada per un expert en *Lean-Manufacturing* que ha repartit les estacions de treball amb dues línies amb forma de U i en flux continu (en la mesura del possible), una forma molt optimitzada que redueix els temps morts, i els moviments inútils dins la fàbrica.



Il·lustració 2 - Flux de treball fàbrica

3.2.2. Estacions de treball



Il·lustració 3 - Distribució de WS

• WS 1: Segellat FF i etiquetatge de cartutxos.

• WS 2: Muntatge de filtres i *frits* (un altre tipus de filtre) i FF CV.

• WS 3: Soldadura per ultrasons del cos de la TC i la coberta.

• WS 4: Inserció manual de la TC.

• WS 5: Muntatge de les membranes i assemblatge de la cambra de purificació.

• WS 6: Segellat del FF de la cambra de purificació.

• WS 7: Muntatge de filtres i CV dels filtres.

• WS 8: Muntatge i inserció BB.

• WS 9: Dispensació de *PPMixes*.

• WS 10: Assecat de *PPMixes*.

• WS 11: Segellat de FF de la RC.

• WS 12: Segellat de film d'alumini de reservoris.

• WS 13: Anàlisi amb CV de les RC.

• WS 14: Muntatge de la DCC:

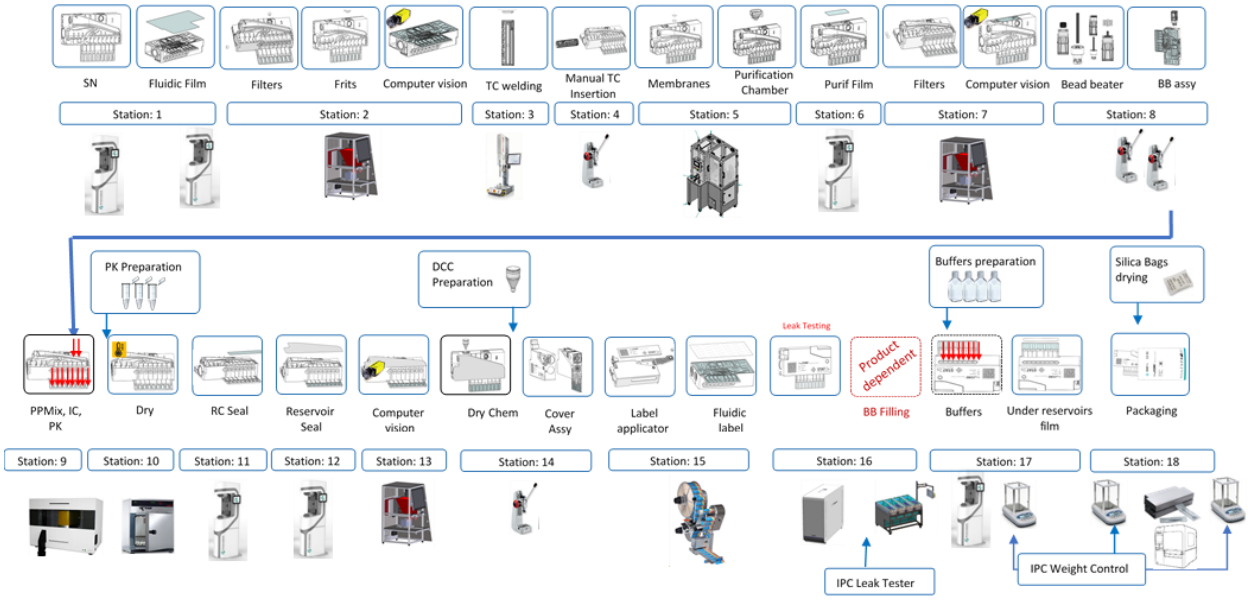
• WS 15: Etiquetatge.

• WS 16: IPC Prova de fugues.

• WS 17: Segellat de film d'alumini de la part posterior dels reservoris i dispensació de *Buffers*.

• WS 18: Envasat dels cartutxos.

En resum:



Il·lustració 4 - Procés de Manufactura per estacions

3.3. Qualitat

Pel que fa la qualitat del producte, és un factor crucial ja que s'està parlant d'un sistema de diagnòstic que ha de ser fiable per que es pugui utilitzar amb seguretat de que el resultat que està donant la prova és correcte. La Qualitat està molt lligada al *scrap*, ja que és la que el genera, i si estudiem aquest *scrap* podem millorar encara més la qualitat del producte si ens dediquem a evitar-ho i a més a més després estudiar-ho.

Per tal de garantir aquest sistema de qualitat, el producte té un seguiment durant la producció en el qual es comproven diferents paràmetres del producte, com l'estanqueïtat del cartutx després del procés de segellat, o si té fugues a l'hora de funcionar. Aquest control es fa com part de la línia de producció i ho podem veure amb les estacions de

treball WS02 (CV), WS07 (CV), WS13 (CV), WS16 (IPC test de fugues) i WS17-WS18 (IPC control de pes), en les quals es detecten possibles errors de producció abans que el cartutx surti de la línia empaquetat.

A més a més dins la línia també s'utilitzen altres eines i mètodes per evitar alguns errors que puguin arribar a produir *scrap*, com el Sistema Lean que implementa algunes de les seves eines a la línia, *kanbans* (cartells visuals que faciliten la identificació de les coses dins la línia i ajuda a mantenir l'ordre), *poka-yoke* (els enginyers dissenyen eines que només permeten fer el procés de forma correcta, cosa que ajuda a disminuir l'error humà), entre altres.

D'altra banda l'equip de control de qualitat té establert unes proves per garantir que tot el lot de producció compleix amb les especificacions del producte i pot ser declarat com a bo, i per tant, pugui ser comercialitzat.

4. Problemàtica

4.1. Registre del *scrap* actual

Actualment el rebuig de la línia (*scrap*) és reportat per l'operari que es troba el defecte en qualsevol de les estacions de treball, aquest quan ho fa, ha d'aturar la feina que estigui fent i disposar-se a anar al *Batch History Record* (Registre de producció) on hi ha un annex en el que es registra aquest *scrap*, s'apunta de forma manual on l'operari escriu el SN (*Serial Number*), diu a quina WS ha hagut de descartar el cartutx, i afegeix una descripció de l'error; després de tota aquesta feina que li pot costar uns minuts l'operari ja pot tornar a les seves taques i seguir a la línia de producció.

Després de la feina de l'operari, els enginyers de la línia de producció recullen aquest registre de *scrap* i el classifiquen amb quins errors s'han trobat, i els introdueixen a una base de dades la qual permet analitzar la informació que aporta aquest registre i saber quins són els errors més freqüents durant la setmana

4.1.1. Què és el *Scrap*

Per poder estudiar el *scrap* de la línia s'ha de conèixer quina és la situació actual sobre la gestió d'aquest, plantejant com es gestiona actualment, s'estudiaran els paràmetres de gestió de el *scrap*.

El *Scrap* és el terme que s'utilitza en l'entorn de producció i *Manufacturing* per a referir-se al rebuig de producte en la línia de producció, un problema que es troba present en qualsevol procés productiu el qual és regit per unes certes exigències de qualitat que s'han de complir.

Cal remarcar la importància que té l'eficient gestió d'aquest fenomen, ja que en qualsevol línia de producció el *scrap* és el que provoca un encariment del producte final. Aquest increment és degut al desaprofitament de recursos i matèria bruta que estava en condicions òptimes per a la fabricació, però que per un error o defecte en la producció s'ha hagut de descartar. Per tant es pot arribar a la conclusió que millorar la gestió del *scrap* pot desencadenar una millora del cost del producte aconseguint abaratir-ho i aconseguir majors beneficis.

4.1.1.1. Causes del *scrap*

Com ja s'ha explicat el *scrap* és degut a la fabricació d'un cartutx que no compleix amb les normes de qualitat que dicten les especificacions del cartutx, les quals garanteixen que el producte podrà complir la seva funció sense donar errors de funcionament.

Per detectar-ho, hi ha establertes algunes estacions de treball que tenen com a finalitat detectar qualsevol error de fabricació o defecte de material que causi *scrap*, abans que el cartutx estigui finalitzat. Aquestes estacions que detecten el *scrap* són:

- WS2 *Filters & Frits Assembly & FF CV*
- WS7 *Filters Assembly & PF CV*
- WS13 *RC, Reservoirs CV*
- WS16 *IPC Leak Test*
- WS17 & WS18 *Weight Control*

4.1.1.2. Classificació del *scrap*

Amb tots els possibles causes que fan aparèixer rebuig a la línia es pot classificar i agrupar les possibles fallades que hi ha, en diversos grups.

Per una banda es classificaran per grups d'estacions (DS), que són punts escollits per fer el registre del *Scrap*, agrupen les WS que estan juntes i engloben un sub-procés en el qual hi ha un control de qualitat al acabar de passar per aquest grup d'estacions. Per tant, a arrel dels 8 IPC que hi ha al llarg de la línia de producció, apareixen les 8 estacions de rebuig que agrupen les 18 WS.

A part, dintre de cada grup d'estacions de treball hi haurà el llistat de possibles errors que es troben en aquests DS, que poden ser fruit d'una de les WS que pertanyen a la DS, a aquests possibles errors els reben el nom de MFM (*Manufacturing Failure Mode*).

DS01: from WS01 to WS02			
MFM001	Bad sealing of qPCR FLUIDIC NETWORK FILM	MFM009	Frit damaged during insertion
MFM002	Missing component: qPCR FLUIDIC NETWORK FILM	MFM010	Frit improperly inserted
MFM003	Misalignment of qPCR FLUIDIC NETWORK FILM	MFM011	Missing component: Filter
MFM004	Double sealing of qPCR FLUIDIC NETWORK FILM	MFM012	Filter damaged during insertion
MFM005	qPCR FLUIDIC NETWORK FILM damaged	MFM013	Filter improperly inserted
MFM006	Cartridge damaged during manufacturing	MFM014	Missing component: Vent
MFM007	Foreign particle in qPCR FLUIDIC NETWORK FILM	MFM015	Vent damaged during insertion
MFM008	Missing component: BB frit		
DS02: from WS03 to WS07			
MFM017	Missing component: membrane	MFM022	Double sealing of Purification FILM
MFM018	Membrane improperly inserted	MFM023	Purification FILM damaged

MFM019	Bad sealing of Purification FILM	MFM024	Foreign particle in Purification FILM
MFM020	Missing component: Purification FILM	MFM078	Double membrane
MFM021	Misalignment of Purification FILM	MFM022	Double sealing of Purification FILM
DS03: WS8			
MFM025	Missing component: BB rotor	MFM027	Lipseal improperly assembled
MFM026	Missing component: lipseal		
DS04: WS 09			
MFM028	PK not dispensed	MFM034	Double drop of PK
MFM029	IC not dispensed	MFM035	Double drop of IC
MFM030	PPMix not dispensed	MFM036	Double drop of PPMix
MFM031	Less quantity of PK	MFM037	Ppmix dispenser tray mispositioned
MFM032	Less quantity of IC	MFM038	Cartridge mispositioned in Ppmix dispenser
MFM033	Less quantity of PPMix	MFM077	Bad dispensed PPMixes
DS05: from WS 10 to WS 13			
MFM039	Bad sealing of RC FILM	MFM045	Bad sealing of Reservoir FILM
MFM040	Missing component: RC FILM	MFM046	Missing component: Reservoir FILM
MFM041	Misalignment of RC FILM	MFM047	Misalignment of Reservoir FILM
MFM042	Double sealing of RC FILM	MFM048	Double sealing of Reservoir FILM
MFM043	RC FILM damaged	MFM049	Reservoir FILM damaged
MFM044	Foreign particle in RC FILM		
DS06: from WS14 to WS16			
MFM050	Pneumatic IPC FAIL		
DS07: WS17			
MFM051	Dry Cartridge Weight IPC FAIL	MFM064	Double sealing of Under reservoir FILM
MFM052	Spill of buffer R1	MFM065	Under reservoir FILM damaged
MFM053	Spill of buffer R2	MFM066	Filled Cartridge weight IPC FAIL
MFM054	Spill of buffer R3	MFM068	Cartridge not weighed in Station 1
MFM055	Spill of buffer R4	MFM069	Cartridge not weighed in Station 2
MFM056	Spill of buffer R5	MFM070	Cartridge not weighed in Station 3
MFM057	Spill of buffer R6	MFM071	Dry Cartridge Weight Over Threshold
MFM058	Spill of buffer R7	MFM072	Dry Cartridge Weight Under Threshold
MFM059	Spill of buffer R8	MFM073	Filled Cartridge Weight Over Threshold
MFM061	Cartridge mispositioned in Buffer filling	MFM074	Filled Cartridge Weight Under Threshold
MFM062	Bad sealing of Under reservoir FILM	MFM075	Spill of Buffers
MFM063	Misalignment of Under reservoir FILM		
DS08: WS18			
MFM067	Weight IPC in Station 3 FAIL	MFM076	Barcode reader error

Taula 1 - Classificació i Distribució de MFM

4.2. Anàlisi de la gestió del *scrap* actual

Actualment, com s'ha comentat en el punt anterior, el *scrap* s'anota a mà per un operari, segons es trobi errors en els cartutxos, i els apunta tota la informació en un paper. Si s'analitza tot el que l'operari ha de fer, i per tant s'ha de tenir en compte per registrar un *scrap* a la línia, es podrà veure quins punts forts i quines carències té el sistema de registre actual.

4.2.1. Procediment de registre

1. S'ha de detectar l'error gràcies als IPC i les CV.
2. S'ha de deixar la feina que està fent, i dirigir-se cap el lloc on estan les fulles de registre del *scrap*.
3. S'han d'agafar les fulles, un bolígraf, el cartutx, i buscar la informació d'aquest (Lot, SN, Producte, hora, data, estació on ha passat i motiu del *scrap*)
4. Un cop es té tota aquesta informació al abast, s'escriu cadascuna de les coses que hem llistat en el punt anterior, a mà.
5. Un cop el *scrap* ha estat registrat, el cartutx pot ser descartat, i l'operari pot tornar a fer la seva feina.
6. Aquestes fulles amb les dades de *scrap* es queden a fàbrica, i fins que no s'acabi el lot no es puja al despatx d'enginyeria per que puguin ser estudiades.
7. Els mateixos enginyers han de dedicar-se a llegir totes les fulles escrites a mà i enregistrar-la a la seva base de dades.

4.2.2. Característiques del sistema de registre

Si anem als punts que són concrets i propis del sistema actual de registre (analògic), podem fixar-nos en el que comporta cada pas que és necessari per fer aquest registre:

- **2:** Quan l'operari ha d'anar a per les fulles de *scrap*, ha de sortir del seu lloc de treball i anar a per l'única fulla que hi ha per anotar el *scrap*, en aquest cas ens podem trobar un conflicte en que dos operaris han detectat un error en dos punts de la línia de producció diferent, i ambdós van a registrar-ho al mateix moment. També s'ha de tenir en compte que és una fulla de paper que és vulnerable a mullar-se, trencar-se o embrutar-se.
- **3:** Per començar, al ser unes fulles que estan soltes poden acabar parant a qualsevol punt de la fàbrica, i inclús a arribar a perdre's, el mateix passa amb el

bolígraf que pot perdre's o que s'acabi la tinta. D'altra altra banda, l'operari ha de buscar tota la informació per si mateixa, i emmagatzemar-la temporalment al cap per poder després escriure-la a les fulles de registre.

- **4:** Quan aquesta informació s'escriu, l'operari ha de tenir la informació molt present per apuntar-la i no equivocar-se sense perdre temps. Aquesta condició introdueix un cert nivell d'error humà, ja que es deixen moltes coses a escriure sota la responsabilitat de l'operari, tot i que aquest pot assegurar-se'n repassant el que ha escrit (tot i que estaria gastant més temps per fer aquest registre). Una avantatge d'escriure-ho a mà, és que es pot indicar tota la informació que l'operari consideri necessària, cosa que pot ajudar després a estudiar aquest *scrap*.

- **6:** El registre de tot el lot es queda a la fàbrica fins que aquest no es doni per finalitzat i llavors pugui ser entregat al departament d'enginyeria per poder estudiar-ho. Durant la fabricació, aquesta informació és poc accessible, ja que per entrar a dins de la fàbrica l'enginyer ha de gastar una bona estona vestint-se adequadament per accedir a la sala blanca, després d'això podrà entrar i tenir accés a les fulles en cas que fos urgent.

- **7:** Per últim aquestes dades enregistrades en un paper, han de ser passades a una base de dades per que puguin ser processades i fer gràfics per obtenir més informació d'aquestes dades. L'enginyer, ha de dedicar una bona estona en entrar una per una totes les entrades del registre de *scrap* al ordinador, feina que duu una quantitat considerable de temps. A més a més, és una feina duplicada, que ha fet primer l'operari d'escriure-la tota, i després l'enginyer ha de tornar a escriure-la a la base de dades.

5. Solució

5.1. Sistema de registre digital

Després d'haver estudiat el sistema de registre actual (analògic) que es fa a mà, s'han detectat alguns inconvenients, que són els que s'han de corregir per tal de millorar el procés de registre.

Per tant, s'ha comparat el sistema de registre digital amb l'analògic, per poder veure quines carències poden cobrir-se amb aquest nou sistema, la taula a continuació resumeix les característiques d'ambdós i es classifiquen amb tres colors en funció de quina característica és mencionada, una positiva (verda), neutre (groga) o bé negativa (vermella).

Sistema digital	Sistema analògic
Registre únic	Registre en diferents fulles
Registre simultani	Registre simultani no possible
Registre ràpid	Procés d'escriptura a mà lent
Informació automatitzada	Buscar tota la informació per si mateix
Baix error humà	Error Humà
Dades accessibles (possibilitat de connectar a la xarxa)	Dades poc accessibles abans d'acabar el lot
Tractar les dades molt més ràpid	Registre per duplicat, perd temps
Procés fàcil d'estandarditzar	Fàcil d'estandarditzar
Depèn de l'electricitat i de la xarxa	Paper (vulnerable, poden perdre's)
Dades poc flexibles	Informació flexible
Manteniment	Informació completa
Actualitzacions lentes	Actualitzacions documentals

Taula 2 - Diferències Sistema de registre manual i digital

Comparant les característiques del actual i les característiques que pot oferir un informàtic, és evident veure que per a un sistema de registre és millor utilitzar un el

digital; ja que els punts a millorar del sistema actual, que és la lentitud del procés de registre i el temps que triga en transmetre les dades i poder ser estudiades pels enginyers (més la feina afegida de registrar tota la informació manualment a la base de dades.

5.2. Requeriments del sistema digital

Comparant els dos sistemes de recollida de dades s'ha de mirar de renunciar mínimament a la qualitat d'informació que dona el registre manual, però aferrant-se a les avantatges que ofereix un sistema informàtic parametritzat que és molt més ràpid. Per tant la solució haurà de tenir les següents especificacions:

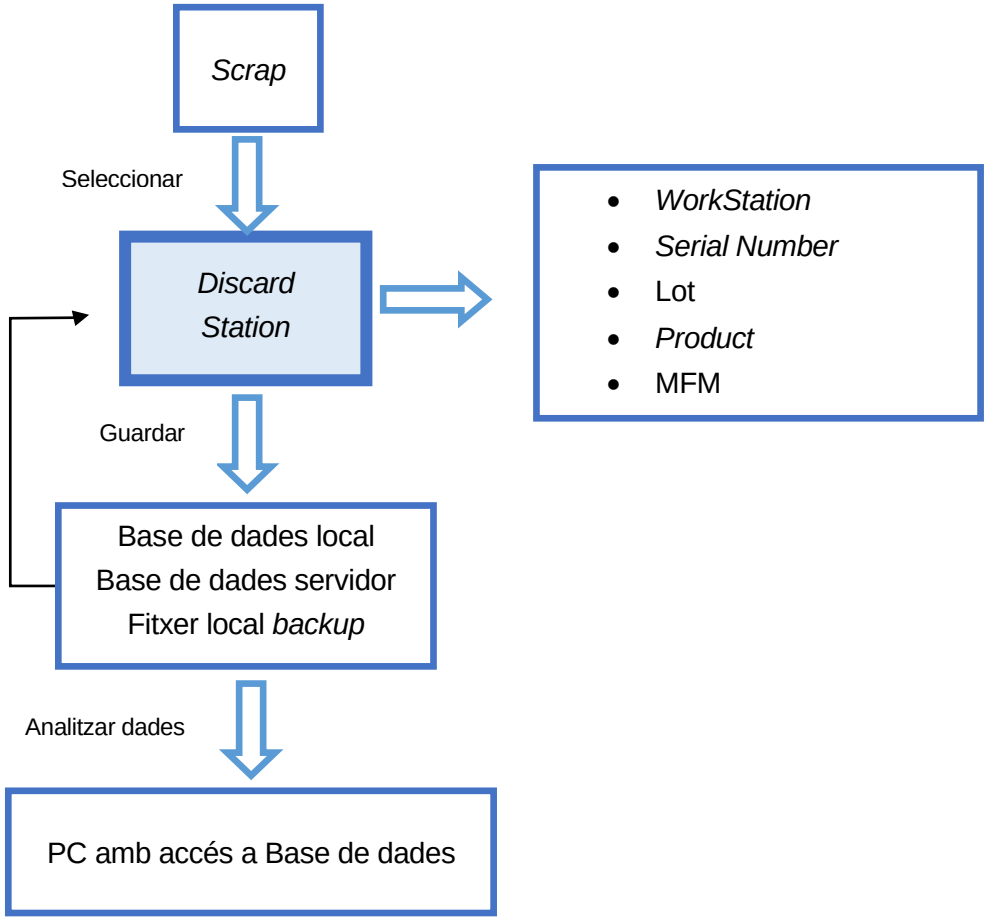
- Fàcil de registrar, un teclat o bé una pantalla tàctil que sigui senzill d'utilitzar.
- Panell intuïtiu, que permeti la menor possibilita d'error possible (poka-yoke a poder ser).
- El dispositiu ha de poder estandarditzar-se per a diferents punts de registre.
- Ha de ser de mida reduïda ja que la línia de producció és petita.
- Ofereixi selecció d'estacions de treball/màquines per saber on ha estat identificat el rebuig.
- Dintre de cada WS es pot escollir quin error ha estat el causant del rebuig.
- També permeti registrar quin ha estat el cartutx dolent, ja sigui escrivint el número de sèrie digitalment o escanejant el *DataMatrix* de cada cartutx. Amb aquesta informació saber a quin lot quin número de sèrie, i de quin producte es tracta.
- Ha d'oferir traçabilitat i per tant registrar la data i la hora.
- Que tingui bona connectivitat i sigui fàcil de comunicar-se amb altres dispositius per tal de poder enregistrar les dades en base de dades.
- Ha de poder comunicar-se a una base de dades d'on tregui la informació que mostra, i que per tant estigui actualitzada sempre, ja que així es pot actualitzar el propi programa des de la base de dades.
- Preu competitiu, sobretot a l'hora de fer el prototip.

Tenint en compte les característiques del sistema digital, s'ha de pensar com implementar-ho, és a dir, pensar quina serà l'estructura del sistema de registre i així poder estudiar cada part per tal que en conjunt funcionin adientment, per posar en marxa

aquest sistema es podrà fer amb una aplicació, una *WebApp*, etc. També s'ha de saber com s'implementarà, sabent quin llenguatge de programació s'utilitzarà per a cada part de l'estructura del sistema de registre.

5.3. Estructura Sistema Digital

Per dissenyar i crear el sistema digital de registre, s'ha de definir una estructura clara, que mostri tots els components i el flux d'informació que travessarà el sistema. A continuació es mostra un esquema de l'estructura que hem mencionat:



Il·lustració 5 - Esquema de l'estructura del sistema digital

Com es veu en l'esquema anterior a l'estructura del sistema de registre hi ha 4 etapes, que són:

- ***Scrap*:** És el motiu del sistema de registre que s'està dissenyant, i per tant l'estímul que fa que el sistema cobri sentit. Es detecta un error en un dels cartutxos, i és d'on prové la informació que interessarà enregistrar, MFM, lot, SN,

WS i producte.

- ***Discard Station*:** Cadascuna de les estacions de registre de *Scrap* que estaran a la línia de producció, aquestes hauran d'estar previstes de tots els recursos necessaris per poder completar la selecció de tota la informació que es vol enregistrar. Aquesta estació de registre és la que es dissenyarà i crearà des de zero, haurà de poder guardar aquesta informació de la forma més òptima possible. A més a més, estarà connectada a la xarxa i es comunicarà amb la base de dades per mostrar i enregistrar la informació en aquesta.

- **Base de dades:** És la plataforma en la que es guardaran les dades seleccionades per tal que quedin registrades, tant en una base de dades local (de l'estació de registre), com en un fitxer local que guardi tota la informació. Alhora també es registrarà la informació seleccionada en una base de dades que es troba en un servidor online, que permetrà que s'hi pugui accedir des de qualsevol dispositiu amb connexió a base de dades i també el sistema de gestió de base de dades MariaDB.

- **PC:** L'ordinador des del qual es tindrà accés instantani a les dades de registre de *scrap* del lot de producció que s'està fent en el mateix moment, aquest es el punt de sortida d'informació, i és el que utilitzaran els enginyers per estudiar el rebuig de la línia i poder millorar-ho.

6. Disseny del sistema de registre informàtic

El sistema digital que es vol implementar té la necessitat de crear una interfície gràfica des de la qual es pugui interactuar amb un hardware adient, i es pugui dur a terme el registre del *scrap*. Aquest punt és el que engloba el projecte en sí; que és crear un programa informàtic que sigui capaç de satisfer totes les necessitats que hi ha a l'hora de fer el registre de *scrap* de la línia de producció de QIAstat-Dx®. Per tant, en aquest apartat s'explicarà com s'ha desenvolupat aquest, quines eines s'han usat per poder crear-ho i com funciona.

Per tant es passarà a explicar quin llenguatge de programació s'ha utilitzat per desenvolupar el programa informàtic de registre, com aquest interacciona amb una base de dades, i que per tant també s'ha de programar en llenguatge de base de dades, i gestionar aquestes.

6.1. Python, Llenguatge de programació

El programa ha estat desenvolupat amb el codi de programació Python, la versió 3.7. És un llenguatge fàcil d'aprendre i molt intuïtiu ja que funciona amb objectes de diferents classes, cadascun amb les seves pròpies funcions. A més a més compta amb un gran ventall de llibreries, aquestes són paquets de funcions estructurades i lligades entre si que permeten que utilitzis funcions sobre un camp específic (com podria ser les matemàtiques, amb operacions o equacions ja calculades prèviament, o alguna que et permeti crear una aplicació, fer gràfics, etc). A més a més compta amb llibreries per treballar amb SQL, que com explicarem més endavant, és el llenguatge de programació que s'utilitza per programar base de dades, cosa que serà indispensable per que el sistema pugui treballar amb aquestes.

D'altra banda, a part del programa que s'encarrega de tractar les dades i agafar i guardar la informació, això s'ha de poder implementar amb l'ajut d'una interfície gràfica que permeti a l'usuari del dispositiu de registre seleccionar el que vol de manera fàcil i intuïtiva. Aquesta és la part amb la que s'ha hagut de tenir més cura, ja que és el que es veu al final i és la plataforma amb la que s'interacciona, i ha de ser prou potent per poder tenir les funcionalitats que ens calen.

6.1.1. Kivy, Interfície gràfica d'usuari

La interfície gràfica d'usuari, que en el món de la informàtica i programació es coneix com GUI, és un programa informàtic que actua d'interfície d'usuari, es basa en l'ús d'un conjunt d'imatges i objectes gràfics, que permet a l'usuari que els pugui veure representats en un pantalla. S'utilitzen per proporcionar un entorn visual en el qual

l'usuari pot comunicar-se amb el programa informàtic .

Escollir una interfície gràfica ha estat difícil, ja que n'hi ha varies i cadascuna té les seves avantatges i els seus inconvenients, tot i que finalment s'ha escollit utilitzar Kivy degut a les avantatges que oferia respecte les alternatives que hi ha.

Kivy és un una interfície gràfica per Python de codi obert i multiplataforma, que permet desenvolupar aplicacions amb interfície d'usuaris molt fàcils. Tot això des d'una eina intuïtiva, per tant, és una bona opció a l'hora de generar prototips de manera ràpida i amb dissenys eficients que poden ser implementats en diversos dispositius gràcies a ser multiplataforma, algunes de les característiques més importants són:

- Kivy ha estat desenvolupat utilitzant Python i Cython, es basa en OpenGL ES 2 i suporta una gran quantitat de dispositius d'entrades, com ara teclats, ratolins, escàners, pantalles tàctils, etc.

- Aquesta interfície està equipada d'una extensa biblioteca de widgets que ajuden a afegir múltiples funcionalitats, són la base del funcionament d'aquesta GUI, i permet tractar els elements gràfics com objectes, que tenen propietats, i els pots col·locar com més adient sigui.

- Com s'ha explicat, Kivy és multiplataforma, pot generar codi que es pot utilitzar en aplicacions orientades a Windows, Linux, Android, OS i Raspbian.

- Compta amb una amplia documentació, que permet aprendre a utilitzar-la, i també amb diferents exemples que mostren l'ús d'aquest GUI .

- Està orientada a la creació de Jocs, cosa que fa que tingui una estètica molt més cuidada que els seus competidors, i molt més fàcil de personalitzar.

- També compta amb un arxiu de suport o disseny kivy (extensió .kv), que té un llenguatge de programació propi i serveix per complementar el codi en python, i dissenyar els objectes que es vulguin des d'allà de forma més senzilla, tot i que l'ús d'aquest arxiu es pot ometre, i només fer servir el propi arxiu de python.

6.1.1.1. Funcionament de Kivy

El Kivy és una GUI orientada a la creació d'apps, per tant s'ha de començar creant un arxiu de python des del qual crea l'App i des d'on es crea el programa que es vulgui fer, amb totes les funcions i classes de python que interessi. D'altra banda, es pot crear l'arxiu de kivy que hem mencionat abans i des del qual es pot dissenyar i estructurar l'aplicació del programa en sí. Aquest funciona organitzant cada objecte o widget amb tabulacions, on l'element que queda més a l'esquerre és el pare, i a la dreta queden els seus fills que depenen de l'element pare, i del qual formen part. Per exemple dins d'un objecte pantalla

(pare) hi podem afegir amb tabulacions widgets (els fills) com etiquetes o botons dins d'aquesta pantalla, cosa que farà que quan el programa mostri la pantalla, pugui mostrar tots els fills que té aquesta, i alhora pugui també mostrar els fills d'aquests altres widgets afegits, i així successivament. D'aquesta manera acoblant uns widgets amb altres, s'aconsegueix crear widgets més complets, i una estructura del programa més elaborada.

Per interaccionar i fer que la interfície gràfica canviï d'estat o canviï la pantalla, la GUI funciona amb un sistema d'esdeveniments, que van associats a accions del programa (per inicialitzar-ho per exemple), però també permet que aquests esdeveniments tinguin lloc després d'interactuar amb l'usuari mitjançant els widgets, ja sigui amb botons, caixes de text, o qualsevol objecte al que se li assigni una acció al tocar-ho o seleccionar-ho. Amb aquesta idea d'esdeveniments el programa té un bucle que es va repetint i va mostrant els canvis que succeeixen després de cada un.

6.1.1.2. Widgets

És molt important entendre què són els widgets i quins tipus hi ha per poder entendre el funcionament, i com s'ha creat el programa.

Com s'ha comentat abans breument, un widget és una classe de python que està inclosa a la llibreria de Kivy, aquesta es comporta com una classe comuna, té els seus atributs i les seves propietats, però a més a més té la capacitat d'interaccionar amb l'usuari causant esdeveniments. Cal remarcar que per utilitzar-los cal importar-los prèviament de la llibreria de kivy a la que corresponguin. Les principals característiques són:

- **Esdeveniments conduïts**: La interacció del programa amb els widgets es construeix sobre els esdeveniments que passen. Si una propietat del widget canvia, aquest pot respondre al canvi si té assignada una funció que estigui relacionada a la interacció amb aquest widget.

- **Separació de preocupacions (entre el widget i la seva representació gràfica)**: El widget es crea i se li assignen les propietats i funcions que siguin necessàries i adients, però pel que fa la seva representació gràfica, és a dir, com es mostra aquest a la pantalla i com el veiem (mida, color, posició, etc) , es crea una pròpia representació gràfica per a cada widget fora de la seva classe, això millora l'eficiència a l'hora d'executar l'aplicació. Per configurar la representació gràfica, tots els widgets disposen del seu atribut c*anvas*, que conté tots els paràmetres gràfics que es poden configurar del widget, que són la mida, la posició, l'altura l'amplada, el color, tipus de lletra, etc.

- **Caixa de col·lisió**: El propi widget quan rep una col·lisió (una interacció amb el widget), aquest reacciona i permet que un esdeveniment tingui lloc com a conseqüència d'aquesta col·lisió, també permet saber si dos widgets han tingut

una col·lisió entre si, aquest característica és la que permet que la interfície gràfica pugui interaccionar amb l'usuari i amb si mateixa per funcionar.

6.1.1.3. Tipus de Widget

Els widgets són els elements de la interfície gràfica d'usuari que formen part de l'experiència d'usuari, és a dir, amb la que l'usuari interacciona . El mòdul *kivy.uix* conté classes per crear i gestionar widgets.

Els widgets de Kivy es poden classificar de la següent manera:

- Widgets UX: Engloba els widgets més clàssics i més comuns, que es poden acoblar entre ells per crear-ne de més complexes. Els UX són els següents:

  Label, Button, CheckBox, Image, Slider, Progress Bar, Text Input, Toggle button, Switch, Video.

- *Layouts*: Els *layout* (disseny), actuen com suport pare per organitzar els widgets fills que té, segons convingui. Aquests són els següents:

  Anchor Layout, Box Layout, Float Layout, Grid Layout, PageLayout, Relative Layout, Scatter Layout, Stack Layout.

- Widgets UX Complexes: Són una varietat de widgets que estan creats a partir de UX clàssics (els mencionats abans), aquests són:

  Bubble, Drop Down List, FileChooser, Popup, Spinner, Recycle View, TabbedPanel, Video Player, VKeyboard.

- Widgets de comportament: aquests no fan cap representació, sinó que actuen en les instruccions gràfiques o en el comportament d'interacció (tàctil) dels seus fills. Els dos widgets de comportament són:

  Scatter, Stenciel View

- Gestor de pantalla: gestiona les pantalles, permet controlar quan canvia d'una a una altre i també les transicions entre aquestes. D'aquest tipus de widget, només hi ha un que es el propi *Screen manager*

Per entendre millor el programa que s'ha creat per a dur a terme aquest projecte convé explicar el funcionament d'algun d'aquests widgets, els més utilitzats, o els més rellevants per a la programació de l'aplicació.

***Label*:** Les Labels, o etiquetes, són widgets que serveixen per representar un text dins d'un requadre (la pròpia label), aquest widget té mètodes per entrar-hi text, el tipus de text que es posa, la posició d'aquest, etc. A part compta amb els atributs que tenen tots els widgets bàsics, que són els de posicionament d'aquest respecte el *layout* en el que es troben.

***Button*:** És semblat a l'anterior, el *Button*, o botó, es un widget igual que les

Labels, l'única diferencia que té amb l'etiqueta és que aquest actua com un botó (com el seu nom indica) i al pitjar-ho es produeix una animació de l'etiqueta per veure que s'ha pitjat; després de clicar es pot indicar que comenci un esdeveniment arrel del clic del botó. Pel que fa el tema d'atributs i funcions que té aquest widget, són els mateixos que tenen les etiquetes, afegint alguns que tenen a veure amb la interacció amb aquest. Les tres funcions que destaquen del botó són:

- *on_press*: És la funció utilitzada per cridar esdeveniments o altres funcions just en el moment de prémer el botó. Per exemple, quan volem que una pantalla canviï a la següent al pitjar el botó, s'utilitzarà la funció *on_press*.

- *on_release*: És la funció utilitzada per cridar esdeveniments o altres funcions just en el moment de deixar de prémer el botó. Per exemple, quan interessa que al pitjar un botó aquest canviï color (*on_press*) i que al deixar-ho anar torni al color original (*on_release*)

***Box Layout*:** És el tipus de *layout* més senzill, serveix per organitzar els fills que li assignem dins d'aquest. Els Box *Layout* poden organitzar-se de forma vertical o horitzontal, de manera que a mesura que s'afegeixen widgets nous (fills) a el *layout* aquests es van distribuint segons s'hagi escollit.

Una utilitat molt pràctica dels Box *layout*, és la capacitat de donar-li als seus fills un atribut anomenat *size_hint*, el qual mitjançant un valor percentual permet establir quina part del Box Layout volem que aquest ocupi tant en la coordenada x com en la coordenada y. Gràcies a aquesta facilitat, el Box Layout es converteix en un widget bàsic i molt útil per organitzar als demés que mostren informació o tenen accions assignades, com etiquetes i botons.

***Grid Layout*:** De la mateixa manera que amb el *Box Layout*, el *Grid Layout* també permet organitzar als seus fills dins d'aquest, però amb una diferencia, l'organització. Aquest widget funciona organitzant als seus widgets fills en forma de graella, és a dir, omple una matriu que es defineix amb els paràmetres *rows* (files) i *cols* (columnes), els quals prenen un valor enter i formen una matriu. Un cop creada la matriu es pot triar com es vol que s'afegeixin els widgets fills, de dreta a esquerra, de dalt cap avall, etc.

L'avantatge d'aquest tipus de *layout*, i el motiu pel qual s'ha utilitzat en aquest projecte, és el fet que els widgets que s'afegeixen tenen forma rectangular (de botó) cosa que encaixa perfectament amb el disseny d'una botonera amb totes les opcions a escollir, com ara la llista dels MFM de la WS7. Al tenir aquesta estructura en forma de graella i col·locació automàtica que proporciona, facilita l'organització de tots els botons, ja que aquests no estan afegits en el codi de disseny (l'arxiu kv) un a un, sinó que es creen i s'afegeixen automàticament a la matriu un a un a partir d'una consulta a la base de dades, a mesura que s'afegeixen queden perfectament organitzats gràcies a el *Grid Layout*.

***Popup*:** El widget *Popup* s'utilitza per crear finestres emergents. De manera predeterminada, el *Popup* cobrirà tota la finestra "pare". Quan es creï una finestra emergent, com a mínim s'han de configurar dos atributs, un *title* (dona el nom al widget i és el que es llegirà a la barra superior del *Popup*) i el *content* (el que hi haurà dins del *Popup*). Aquest eina es útil per mostrar missatges com a conseqüència d'un esdeveniment, com per exemple, un error en el que les dades introduïdes no eren correctes.

***Screen Manager*:** Un widget essencial si es vol crear una aplicació multi pantalla, com ha estat el cas del sistema de registre del *scrap*. *Screen Manager* permet gestionar el canvi de pantalles, d'una a una altra aplicant una transició. Per funcionar tan sols cal que s'afegeixin els widgets fills (pantalles) al *Screen Manager*, i a partir d'aquí es podrà canviar de pantalla.

Per fer aquest canvi de pantalla, que és la funció bàsica d'aquest widget, es pot utilitzar l'atribut *current* assignant quina és la pantalla actual en el moment en que s'actualitza l'atribut. Alternativament també es pot utilitzar la funció *switch_to*, que canvia de pantalla en el moment que s'executa la funció.

***Text Input*:** El widget *Text Input* proporciona un quadre per a text normal (com una Label) editable. Les funcions de múltiples línies de text, selecció de text i navegació amb cursor poden ser utilitzades en aquest widget. Aquest widget té varies funcions que han sigut interessants a l'hora de registrar el *DataMatrix* del cartutx, ja que aquest és llegit com una seqüència de molts números que s'han d'introduir. Algunes d'aquestes funcions i atributs que s'han d'entendre per saber com funciona el sistema de registre són:

- *focus*: És un atribut booleà. Posant-ho a *True* sol·licitarà que el teclat, o dispositiu d'entrada de text li faci alguna entrada, posant-ho a *False* s'alliberarà el teclat. Només es pot tenir un widget focus, quan el focus del teclat canviï, automàticament el widget que estava amb estat *focus* en *True*, passarà a *False*. Per tant és un atribut que cal tenir controlat quan es vulgui entrar text.

- *on_text_validate:* Es dispara només quan l'atribut *multiline* (multi línia) és *False* i per tant quan es prem el botó "Entrar", el quadre de text canvia el seu estat de *focus* de *True* a *False*.

6.1.1.4. Mètodes i funcions

Per a la programació del sistema de registre s'han utilitzat moltes funcions, a part de les més comuns de python com la iteració *for*, l'ús de diccionaris i llistes (i les seves funcions) i l'ús de llibreries comuns com *datetime*. A part de les que s'acaben de comentar i les pròpies que tenen els widgets i que també s'han comentat en l'apartat anterior, hi ha

alguns mètodes que val la pena explicar-los degut a la rellevància que tenen vers el projecte.

Herència de classe: Precisament aquest no és un mètode propi i exclusiu de Kivy, sinó que estem parlant de l'herència de classe de python, que és en certa manera com funcionen els widgets i com es criden, un cop posem un widget aquest té les propietats i mètodes de la classe pare. Per exemple quan posem un botó, les propietats del botó que posem són les heretades de la classe Butt on.

L'Herència de classe també ha sigut molt útil per utilitzar informació compartida en diferents pantalles (dues pantalles diferents i per tant dues classes diferents). La informació que es registra en diferents pantalles, ha de quedar emmagatzemada en algun lloc, que a la vegada es pugui consultar des de qualsevol classe que es vulgui, per tant, al utilitzar l'herència, s'ha pogut crear un atribut que conté la informació registrada i es pot escriure i consultar en totes les classes que hagin heretat aquest atribut de la classe pare.

***add_widget*:** Com diu el nom en anglès, significa afegir un widget, i això és el que fa. Amb aquesta funció s'afegeixen tots els widgets que formaran part de la interfície d'usuari que es creï. Característiques d'aquesta funció destacables:

- És molt important dominar l'ús d'aquesta funció, ja que segons l'ordre en que s'afegeixin els widgets, la interfície tindrà un aspecte o un altre.
- Permet afegir widgets des de l'arxiu de python, que és des d'on es pot obtenir la informació de base de dades, per tant és essencial que es pugui utilitzar aquesta funció, ja que des de l'arxiu de disseny no podríem afegir els botons de la forma que es vol per tal que es mantingui actualitzada la informació representada de la base de dades.
- Els widgets afegits tenen un índex, que es pot donar al widget afegit, si es vol que s'afegeixi abans o després que els seus "germans"
- També permet afegir-li *canvas* al widget afegit, per configurar l'aspecte d'aquest abans que sigui afegit o després, segons es defineixi.

Bind: És un mètode que permet lligar un esdeveniment o una propietat a la crida d'una funció, que significa que permet que un objecte pugui enllaçar una funció determinada. Aquest mètode ha sigut molt útil a l'hora d'assignar-li esdeveniments a un botó, o accions a un *popup*.

Cal destacar l'ús de la funció anònima lambda per cridar als esdeveniments que interessen, ja que ha sigut necessari a l'hora d'assignar als botons funcions com canviar de pantalla, o enregistrar un valor, que són funcions que simplement actualitzen una

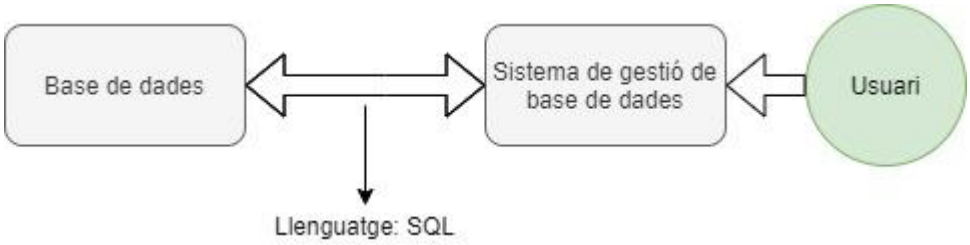
propietat i no retornen res.

6.2. Base de dades

Una base de dades és un conjunt de dades que pertanyen a un mateix context, i queden emmagatzemats d'una manera sistemàtica i sobretot amb un criteri d'ordre, si les dades tenen un ordre podran ser utilitzades posteriorment. Per accedir-hi es fan consultes en les que es llegeix o bé s'escriu en aquesta base de dades.

Les bases de dades per tant són una manera molt bona d'emmagatzemar dades i tenir-les ordenades per una posterior consulta, aquestes característiques, deixen veure que és un sistema molt adient per emmagatzemar les dades obtingudes del sistema de registre del que tracta aquest projecte

Hi ha programes denominats sistemes gestors de base de dades SGBD (que en anglès és *Database Management System* o DBMS), aquests permeten emmagatzemar i accedir de forma organitzada i fàcil.



Il·lustració 6 - Esquema funcionament Base de dades

6.2.1. MariaDB

Maria DB és un sistema SGBD que deriva del conegut MySQL, que utilitza un sistema de gestió de base de dades SQL, que permet escriure i consultar bases de dades.

MariaDB al derivar de MySQL, té les mateixes característiques i funcionalitats que MySQL, però aporta algunes millores en quant a rendiment i optimització de consultes, a més de ser de llicència oberta i ser el sistema que utilitza l'empresa Stat-Dx® per emmagatzemar la seva informació.

Com s'ha dit, MariaDB és un sistema de gestió de base de dades basat en SQL, i per tant caldrà explicar, com funciona i en que es basa aquest llenguatge .

6.2.2. SQL

El SQL es caracteritza per l'ús de l'àlgebra relacional (conjunt d'operacions que descriuen com es manipulen les dades i càlcul relacional (és el llenguatge de consulta de base de

dades), amb aquestes eines s'és capaç de realitzar consultes a la base de dades per recuperar informació emmagatzemada sense fer cap canvi en aquesta.

Les dades amb les que tracta SQL són de quatre tipus, i tenen un format equivalent en la programació de python, cosa que resulta molt convenient. Poden ser del tipus :

- Varchar: Cadena de paraules i símbols alfanumèrics, seria equivalent al string de python.
- Int: Es refereix a números enters amb o sense signe, és equivalent al int de python.
- Date: Una data del calendari en format AAAA/MM/DD, equivalent a la llibreria de python *datetime*.
- Date: Una hora del dia en format HH/MM/SS, equivalent a la llibreria de python *datetime*.

SQL és un llenguatge de programació d'alt nivell, per tant el converteix en un llenguatge amb una alta productivitat en codificació, ja que amb una sola línia es poden realitzar una gran quantitat d'operacions que serien molt més llargues d'executar amb un altre llenguatge de programació de baix nivell.

Per tal d'organitzar la informació, SQL treballa amb objectes de dades, que fan referència a bases de dades, taules per organitzar informació i separar-la en camps dins d'una mateixa base de dades, índexs per identificar cada fila de dades i fer més accessible la taula per mitjà d'índexs, i també disparadors (*triggers*) que enllacen taules, de forma que la relació s'activa després d'algun esdeveniment sobre la base de dades.

Per tal de fer les consultes que convinguin, el llenguatge de programació SQL té diferents operacions, es diferencien en llenguatge de definició de dades (DDL) i llenguatge de manipulació de dades (DML). En tots dos casos és un llenguatge molt intuïtiu, pràcticament el que es programa és una frase que descriu la consulta.

6.2.2.1. Llenguatge de definició de dades

Està orientat a la creació de nous objectes, modificar i eliminar-los, és a dir, s'utilitza per estructurar la base de dades, les operacions que té són les següents:

- CREATE: Permet crear un objecte com una base de dades o una taula.
- ALTER: Si interessa modificar alguna característica d'un objecte existents, com per exemple afegir un camp en una taula o canviar-ho.

- DROP: És la operació contraria al CREATE, s'utilitza per esborrar objectes que ja no interessa que estiguin.

- TRUNCATE: Només s'aplica a taules i serveix per esborrar tot el contingut d'una taula en concret.

6.2.2.2. Llenguatge de manipulació de dades

A diferencia del DDL, el DML està pensat per utilitzar comandes que manipulin directament les dades emmagatzemades dins la base de dades, les operacions més rellevants són:

- SELECT: És una operació que s'utilitza per consultar una taula concreta.

- INSERT: Serveix per afegir informació a la base de dades, pot ser una sola dada o més d'una.

- UPDATE: Quan es vol actualitzar els valor d'un registre fet prèviament s'utilitza aquesta operació.

- DELETE: Serveix per esborrar un registre dins d'una taula.

Per ajudar a escollir la informació que es vol de cada registre determinat i de cada taula, existeixen més comandes per especificar el que es vol consultar, els més bàsics i utilitzats en aquest projecte per fer consultes són els següents:

- FROM: Indica la taula que es vol consultar. Quan ens trobem en el cas que hi ha més d'una taula, i s'ha d'aplicar una condició, és necessari fer una consulta combinada amb una clàusula.

- WHERE: Clàusula que especifica quina condició ha de complir-se per tal que les dades siguin retornades d'acord a la condició especificada. Per donar condicions més complexes es poden utilitzar els operadors lògics OR.

- GROUP BY: Serveix per especificar quina agrupació de dades es vols consultar indicant el nom del camp que es vol consultar.

- HAVING: Té la mateixa funció que el WHERE, aplica una condició, però en aquest cas referint-se al grup de dades especificat en el GROUP BY

- ORDER BY: S'utilitza per definir com es vol que es retornin les dades, si en ordre ascendent o descendent (ordenat alfa numèricament).

7. Implementació

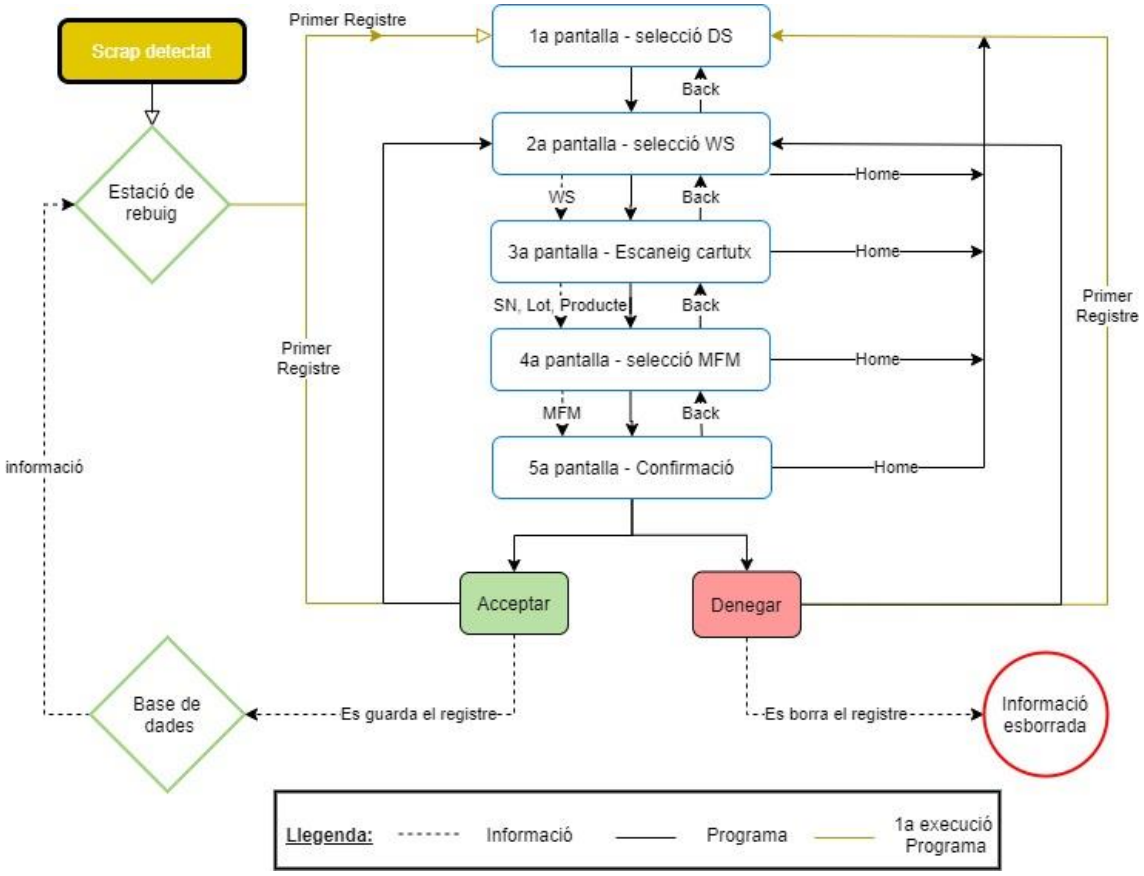
La implementació és el punt en el que el projecte cobra sentit i tota la informació recopilada tant de necessitats per al projecte com eines i mitjans per dur-ho a terme s'apliquen en un sistema que permet registrar el *scrap* de la línia de producció de Stat-Dx. En aquest punt s'explicarà el funcionament del sistema de registre, de com funciona el programa, com interacciona amb la base de dades, i com pot implementar-se tot plegat en un dispositiu que es pugui utilitzar per fer el registre del *scrap*.

7.1. Funcionament

Abans d'explicar detalladament el funcionament del sistema de registre i les parts que el conformen, és interessant descriure el funcionament de tot el sistema en conjunt, i veure quin paper juga cada part en el sistema global.

7.1.1. Descripció del funcionament

Aquest sistema té sentit només quan es produeix un *scrap*, que és l'esdeveniment que conté tota la informació que es vol registrar, per tal que pugui ser estudiat més tard. Un cop s'ha de registrar el rebuig d'un cartutx, l'operari ha de dirigir-se a l'estació de rebuig (DS – *Discard Station*) més propera. Les DS són vuit en total i són les descrites a l'apartat 5.1.1.2 Classificació del *scrap*, cada una conté una o més de les 18 WS que té la línia de producció del QIAstat-Dx®.



Il·lustració 7 - Esquema funcionament App de registre

Un cop l'operari es trobi a la DS és el moment en el que entrarà en contacte amb el sistema de registre, on es trobarà una pantalla tàctil que projecta un mini ordinador que està executant el programa de python, és a dir l'app amb la qual l'operari interactua amb el sistema de registre .

La primera pantalla (DS) que es troba l'operari li mostra una botonera amb les 8 DS per que s'esculli una i quedi registrat des de quina estació s'ha fet el registre. La idea és que hi hagi un dispositiu de registre a cada DS, i que per tant aquesta pantalla només es veurà just després d'iniciar el programa, un cop s'hagi fet el primer registre ja no es veurà més. A part de la botonera aquesta pantalla, i totes les demés tenen una columna a la part dreta de la pantalla, que resumeix la informació registrada (WS,Nº de cartutx, Lot, Producte i MFM) per poder anar seguint el registre en tot moment, a més a més, compta amb dos botons que permeten a l'usuari anar a la pantalla anterior, i un altre que et porta al inici del registre, en el cas que es vulgui començar des de 0 aquest. Després d'haver escollit l'estació de rebuig el programa porta a l'operari a la pantalla següent.

La segona pantalla (WS) és semblant a l'anterior, i mostra una botonera que correspon a les estacions de treball que estan contingudes dins de la DS seleccionada. A part de la botonera, segueix amb la mateixa estructura que la pantalla anterior, i té a la part dreta de la pantalla la mateixa columna anterior amb la informació resumida fins el moment,

que és cap i els dos botons d'enrere, i inici.

La tercera pantalla (*Scan*) és la que s'encarrega de llegir la informació característica del cartutx rebutjat, el Nº de cartutx, lot, i producte. Aquesta informació es troba tant al *DataMatrix* de la SN Label (es col·loca a la WS1), com a la *Cartridge Label* (es col·loca a la WS15), així que depenent en el punt de la producció en el que s'hagi descartat el cartutx, s'escanejarà un *DataMatrix* o un altre. Per enregistrar la informació característica del cartutx, l'operari ha de clicar en el camp corresponent per a l'escriptura i escanejar el *DataMatrix*, un cop fet això, la informació s'escriu a la pantalla i s'enregistra al moment i automàticament passa a la següent pantalla. Igual que les altres pantalles compta amb la columna de la dreta amb la informació enregistrada fins el moment i també els botons enrere i inici.

A la quarta pantalla (FM) es fa l'últim registre, aquest és el de l'error que s'ha trobat en el cartutx, aquesta pantalla compta amb una estructura igual que la primera i la segona pantalla, és una botonera que conté tots els MFM de l'estació de rebuig que s'ha seleccionat prèviament., com s'ha comentat en les altres pantalles també compta amb la columna que resumeix la informació registrada fins al moment i té els dos botons d'enrere i inici. Un cop es selecciona el MFM, aquest queda registrat i passa a la següent i última pantalla.

La cinquena i última pantalla (*Confirm*), serveix per comprovar que el registre ha estat ben fet i tota la informació enregistrada és la correcta. En aquesta pantalla hi ha la mateixa columna que havia aparegut a les anteriors pantalles, i es veu tota la informació ben representada i junta, i dos botons, un que serveix per acceptar el registre i l'altre per denegar-ho. Un cop s'accepta o es rebutja, el programa torna a portar la interfície a la segona pantalla ja que com s'havia comentat la DS no s'ha d'escollir més ja que el dispositiu ha de quedar-se ala DS on s'ha utilitzat.

Per últim al finalitzar el registre, es torna a veure la segona pantalla preparada per fer-ne un de nou.

7.2. App

Tal com es descriu a la documentació de Kivy, el llenguatge de programació d'interfícies gràfiques, aquesta s'utilitza per crear aplicacions, compatibles amb múltiples SO, i amb una llibreria molt completa per crear interfícies d'usuari. La idea del programa o aplicació, és que estigui format per diferents pantalles que permetin escollir la informació que interessa registrar, permetent que a cada una es pugui interactuar de la forma més senzilla i ràpida, i aconseguint que després d'això la informació quedi registrada a la base de dades, i que després torni a estar disponible per fer un altre registre.

No s'ha d'oblidar però, que el programa ha estat desenvolupat amb llenguatge python, per tant s'han utilitzat moltes funcions d'aquest llenguatge i diverses llibreries de python que han estat necessàries per a implementar les funcions més específiques. El programa funciona separat amb tres arxius, el main.py, que conté el cos del programa y defineix tot el funcionament; el main.kv, que és l'arxiu de disseny de kivy i per últim el database.py, que és un arxiu de python amb les funcions que s'encarreguen de consultar i escriure a la base de dades. Per veure el programa sencer i el codi del programa veure L'Annex 1 amb tots els arxius.

Per explicar el funcionament a detall de l'aplicació es separà l'explicació del programa per parts principals. Cal comentar que aquestes parts són Widgets al cap i a la fi, ja que són elements de Kivy. Per tant s'explicaran per separat les següents parts:

- Un element que no apareixerà visualment, i que és l'aplicació en si, el MainWid, un widget que és el *ScreenManager*, i que és el pare de tots els altres widgets que configuren el programa.

- Hi ha un element que és present a totes les pantalles, i és el resum del registre que recull la informació que s'ha anat enregistrant. A més a més compta amb un botó *Back* (que permet anar a la pantalla anterior) i un de *Home* (que fa anar a la primera pantalla).

- La primera pantalla, la segona i la quarta, que són pantalles similars, amb una botonera i una barra lateral amb informació i botons de navegació del programa, és la mencionada en el punt anterior, i s'explicarà a part.

- També hi ha la tercera pantalla, que és la que recopila més informació, la característica del cartutx registrat, i és la que compta amb un lector de codis 2D per efectuar la lectura dels *DataMatrix*.

- Finalment l'última part de la que es parlarà serà l'ultima pantalla que serveix per acceptar o rebutjar el registre efectuat.

7.2.1. Widget principal, MainWid

Aquest Widget és el que farà córrer l'aplicació s'encera, ja que és del Tipus *ScreenManager*, i és la que té la capacitat de canviar d'una pantalla a una altre, que és el que interessa per aquest programa, que a mesura que es fan registres vagin avançant les pàgines.

Aquest widget al ser el principal i pare de tots els demés widgets del programa, donarà en herència totes les seves propietats i funcions, a tots aquells widgets que la rebin, això permetrà que hi hagi un punt en comú de totes les pantalles per tal de tenir comunicació entre elles, serà clau en l'enregistrament d'informació.

7.2.1.1. Propietats

Pel que fa les propietats del widget principal, és només un i és el self.cart, que és una llista de vuit caselles de llargada. Aquesta llista és l'encarregada d'emmagatzemar la informació durant el registre, està pensada per que pugui ser cridada des de qualsevol widget que tingui herència de MainWid. Cadascuna de les caselles està pensada per emmagatzemar una informació que interessa registrar:

- En primer lloc es registre l'estació de rebuig, per tant la primera posició de la llista la ocupa la DS
- Seguint l'ordre de registre el següent element a registrar és la WS on s'ha detectat l'error.
- En tercer lloc es registra el element que permet identificar el cartutx, és a dir, el nº de sèrie o SN.
- Junt amb la informació anterior, es registra en quarta posició el lot de fabricació al que pertany el cartutx, aquest apareix dins del SN, però es separa per que sigui més fàcil d'identificar el lot.
- En la cinquena casella de la llista es registra el MFM escollit per l'operari.
- L'últim element que es registra durant el escaneig, i es fa en sisena posició de la llista, és quin producte s'està rebutjant.
- El setè element guarda el dia en el qual s'ha fet el registre, ja que així queda traçabilitat del *scrap*.
- L'últim i vuitè element és la hora a la que s'ha efectuat el registre.

7.2.1.2. Funcions

Les funcions del MainWid són molt bàsiques, són les funcions que permeten navegar per l'aplicació, i anar d'una finestra a una altra. En totes elles s'utilitza la funció *current* de la llibreria de *Screen Manager* de kivy, que quan s'executa fa que la pantalla que s'està visualitzant en aquell moment sigui la pantalla que ha executat l'ordre *current*.

Totes les següents funcions han estat dins del MainWid ja que així poden ser cridades des de qualsevol altre widget (pantalla) que hagi rebut herència, i així poder canviar la pantalla de qualsevol punt del programa:

- goto_dp(): Funció que permet anar a la finestra per escollir DS, quan aquesta es crida, a part de dur a l'usuari a la pantalla en qüestió, també crea una llista amb

tots els botons de les DS. Per crear la llista ho fa a partir d'una funció del DPWid.

- goto_ws(): Funció que permet anar a la finestra per escollir WS, quan aquesta es crida a part de dur a l'usuari a la pantalla en qüestió, també crea una llista amb tots els botons de les WS, per crear la llista ho fa a partir d'una funció del WSWid. A més a més també crida una funció de InfoWid, que s'encarrega de refrescar la informació representada a la columna de la dreta.

- goto_scan(): Funció que permet anar a la finestra per registrar el cartutx escanejant el *DataMatrix* d'aquest. A més a més també crida una funció de InfoWid, que s'encarrega de refrescar la informació representada a la columna de la dreta.

- goto_fm(): Funció que permet anar a la finestra per escollir FM, quan aquesta es crida a part de dur a l'usuari a la pantalla en qüestió, també crea una llista amb tots els botons dels FM, per crear la llista ho fa a partir d'una funció del FMWid. A més a més també crida una funció de InfoWid, que s'encarrega de refrescar la informació representada a la columna de la dreta.

- goto_confirm(): Funció que permet anar a la finestra final i acceptar o rebutjar el registre efectuat. Per fer-ho primer actualitza tota la informació registrada amb una funció del ConfirmWid (és una copia del InfoWid, ha estat separat degut a diferències gràfiques).

7.2.2. Caixa d'informació, InfoWid

Aquesta part està formada per un *BoxLayout*,, el tipus de *Layout* més senzill de tots, s'encarrega simplement d'organitzar els widgets fills que contenen la informació que es vol representar en aquest widget.

La idea és que aquest part vagi representant un resum de la informació que es va guardant, i després tenir un parell de botons per navegar per l'app.

Aquest widget no té cap propietat destacable, però si que té vuit widgets i tres funcions essencials per que aquesta caixa d'informació i navegació pugui funcionar.

Il·lustració 8 - InfoWid dins la App

Té 8 widgets, que estan distribuïts verticalment al llarg de la columna que forma aquest InfoWid, i són els següents:

- *Image*: És una imatge amb el logotip de l'empresa QIAGEN.
- *Label*: Són 5 etiquetes cadascuna té el seu propi text i són les següents:
    - *Station*: Conté la informació sobre l'estació en la que es detecta el *scrap*.
    - Lot: Diu el lot al que pertany el cartutx rebutjat.
    - *Cartidge* nº: Diu el número del cartutx en qüestió.
    - *Panel*: Mostra el producte al que pertany el cartutx escanejat.
    - *Failure Mode*: Mostra la informació relativa al error identificat en el cartutx.
- Els dos botons, són els que serveixen per navegar, i són els següents:
    - *Home*: Botó que permet anar a la pantalla principal, la pantalla on es registra la DS, per tornar a començar el registre des de 0, ho fa amb la funció home() de InfoWid.
    - *Back*: Botó que permet anar a la pantalla anterior a la que es trobava navegant l'usuari, ho fa amb la funció back() de InfoWid.

7.2.2.2. Funcions

Les tres funcions que té són les que s'han referenciat en el punt anterior, i serveixen per que aquest widget pugui complir la seva funció de mostrar la informació registrada i navegar per l'app. Les funcions són les següents:

- updateinfo(): És una funció molt senzilla, s'encarrega simplement de modificar l'atribut text de de cada una de les etiquetes amb la informació del registre. Per fer-ho canvia directament l'atribut text de cada etiqueta i li afegeix la cadena de strings amb la informació de la propietat mainwid.cart que conté la informació de registre i s'hi pot accedir gràcies a la herència de classe, ja que InfoWid rep herència de MainWid.

- home(): Funció que fa anar a la pantalla d'elecció de DS (la primera), per fer-ho aplica la funció goto_ds() que utilitza de la herència de InfoWid. Quan es va la primera pantalla també s'ha d'actualitzar la informació de la columna de la dreta, per fer-ho, es buida la llista mainwid.cart, i s'aplica la funció updateinfo() amb el que s'aconsegueix que la columna d'informació no mostri cap dada registrada anteriorment.

  Abans però de que es canviï la pantalla i s'esborri la informació, fa falta fer una cosa, que és esborrar els botons que estaven creats, ja que quan es crida la funció goto_ds() es creen els botons amb les DS, i si no s'esborrés, apareixerien tots els botons duplicats, i si es tornés a pitjar, sortirien triplicats, i així successivament.

- back(): Funciona de forma similar que la funció home(), però en aquest cas permet anar a la pantalla anterior a la que s'estava, utilitza la funció goto_ds(), goto_ws(), goto_scan() i goto_fm() segons es trobin a la pantalla de WS, *Scan*, FM o *Confirm* respectivament; totes aquestes funcions seran cridades utilitzant l'herència de InfoWid. Quan es va a la pantalla anterior també s'esborra la informació que s'havia registrat en aquesta abans de passar a la següent pantalla, amb la idea de poder rectificar i poder escollir correctament el que es volia registrar; per fer-ho s'esborra la informació que toca de la llista mainwid.cart i s'aplica la funció updateinfo() per actualitzar la informació mostrada.

  Igual que amb la funció d'abans s'ha d'esborrar els botons generats automàticament a les pantalles DS, WS i FM, ja que quan s'executa la funció corresponent per anar a la pantalla també es creen de nou els botons. Per discernir sobre quins widgets i quina informació s'ha d'esborrar, hi ha una estructura de condicionals que considera en quina pantalla es troba l'aplicació en aquell moment, i esborra la informació i els widgets necessaris que convinguin.

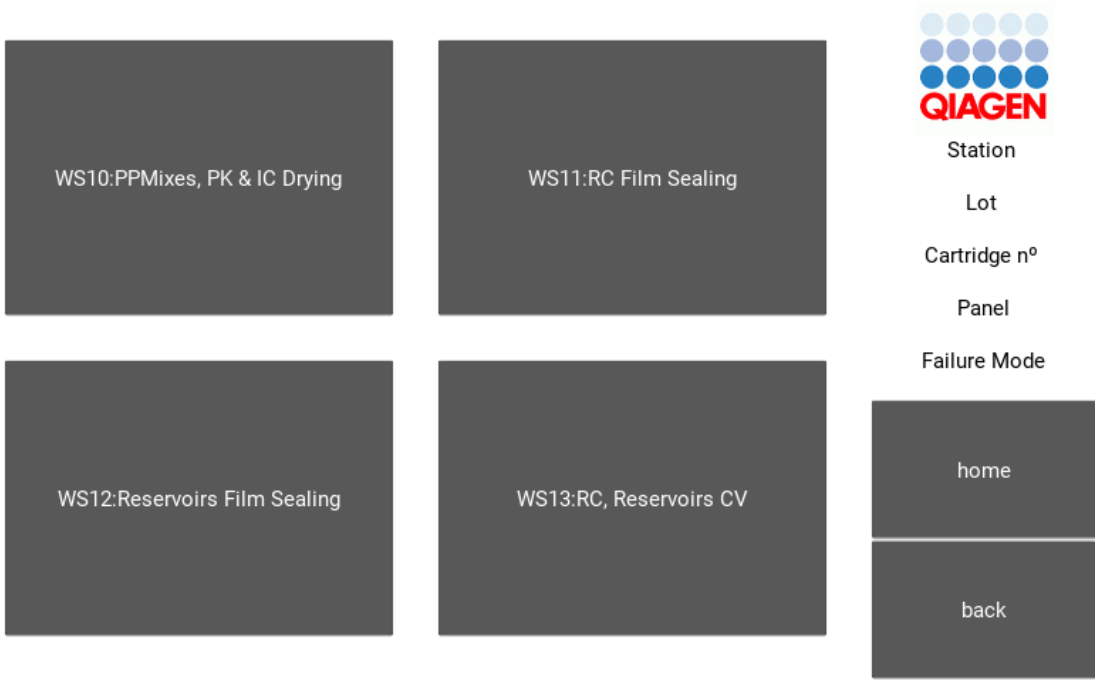
7.2.3. Pantalles DS, WS i MFM

Aquestes tres pantalles segueixen totes la mateixa estructura contenen una botonera (*GridLayout*) i després una columna amb el widget InfoWid, comentat en el punt anterior, que mostra la informació de registre.

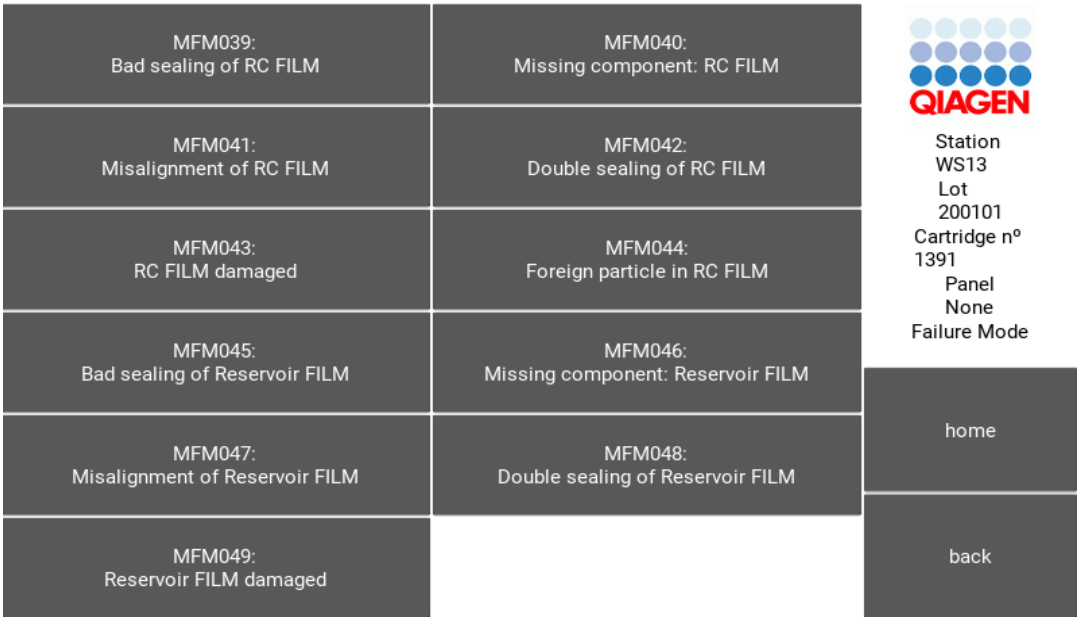
Per aconseguir aquesta distribució, s'ha hagut d'utilitzar un *layout* auxiliar (*BoxLayout*), que ha rebut el nom de MenuWidDS, MenuWidWS i MenuWidFM respectivament, un per a cada pantalla; amb l'ajuda d'aquests widgets auxiliars s'ha pogut organitzar la botonera per una part de la pantalla i la columna amb informació a l'altre.

Com totes tres pantalles segueixen un funcionament comú, s'explicaran les botoneres de totes les pantalles a la vegada. Aquestes botoneres estan contingudes dins del *layout* auxiliar que s'acaba de comentar, i són *GridLayout* que organitzen els widgets fills en forma de matriu, cosa molt pràctica per una botonera.

Il·lustració 9 - Primera Pantalla DS



Il·lustració 10 - Segona pantalla WS (DS5)



Il·lustració 11 - Quarta pantalla (DS5)

7.2.3.1. Propietats

Per tenir controlats els botons que es creen a la *Grid Layout*, s'ha creat una propietat anomenada *list*, que és una llista que emmagatzema els widgets ja creats (objectes).

També s'han configurat les propietats que defineixen el número de columnes i files, per tal que la informació dels botons hi càpiga de forma correcte i es pugui llegir de forma adequada.

- DPWid: Té 3 files i 3 columnes crea una graella de 9 botons, suficientment amples per representar la informació i que càpiguen les 8 DS diferents.

- WSWid: en aquest cas només s'ha configurat el paràmetre de files que ha estat de 3, amb aquesta configuració Kivy distribueix els botons en tres columnes i en el cas de necessitar representar un quart botó es crea una altre columna per que càpiguen més botons. En aquest cas es fa així ja que els noms complets de les WS són molt llargs i no caben dins d'un botó molt petit.

- FMWid: És el cas més complex ja que la quantitat de botons a crear varia molt en funció de la DS en la que l'usuari es trobi, va des de 1 a la WS18 a 24 a la WS17, per poder representar tots els botons s'han fixat només dues columnes, i que Kivy s'encarregui d'organitzar la resta de botons afegint files i reajustant la mida dels botons. D'aquesta manera s'ha aconseguit que en el pitjor cas de representar fins a 24 MFM, es pugui llegir la informació, i encara quedi marge per si s'afegeix alguna de nova.

7.2.3.2. Funcions

Les funcions necessàries per a cada pantalla són anàlogues, hi ha que creen els botons (createDP(), createWS(dp) i createFM(ws)) i també les funcions de selecció de botó (selDP(), selWS() i selFM()), que permeten saber quin botó s'ha pitjat i així poder guardar la informació seleccionada. Per tant s'explicaran aquests dos grups que tenen les tres pantalles en comú:

- Funcions de creació de botons: en els tres casos, la funció *create* s'encarrega de crear els widgets botons que corresponen a les eleccions que hi ha per fer a cada pantalla (DS, WS i MFM). Per fer-ho es recorre un diccionari amb totes les DS, WS i MFM com a clau i una descripció d'aquestes corresponent, al recórrer cada diccionari, es pot crear un botó que contingui el mateix text que el de la descripció del diccionari i també la seva clau que diu el codi de la DS/WS/MFM; a més a més, aquest botons reben el mètode *bind()* per assignar-li una acció a aquest, que serà la funció de selecció corresponent, que s'explica a continuació.

  Per aconseguir aquests diccionaris, es crida la funció alldp(), allws(dp), allmfm(ws) segons el programa estigui representant DPWid, WSWid o FMWid; aquestes funcions pertanyen al arxiu database.py, son programes que comuniquen i consulten la base de dades per poder tenir la informació d'aquesta al moment.

A mesura que es creen els botons es van afegint a la propietat list, que és una llista de llistes, on cada sub llista conté el objecte botó, i el codi de DS/WS/MFM. Alhora es van afegint al corresponent *Grid Layout* els botons, de manera que quan s'acaba d'executar la funció tots els botons estan afegits, i guardats a la llista.

- Funcions de selecció: El propòsit d'aquestes funcions és identificar el botó que s'ha pitjat en el moment de fer la selecció i canviar de pantalla, aquesta funció és la que porten assignada tots els botons creats de cada *Grid Layout* (a cada *Grid Layout* s'executa la funció corresponent a la seva pantalla). Per saber quin ha estat el botó seleccionat, s'utilitza la propietat last_touch que té la classe *Button*, que per defecte té el valor de "*None*", així que, el que es fa és recórrer la llista que conté tots els botons, i es pregunta si es posa la condició de si el segon element de la subllista (l'objecte *button*) té un valor de la propietat last_touch diferent de "*None*", si el valor és diferent d'aquest, vol dir que ha estat el botó pitjat. Quan se sap el botó seleccionat es pot guardar a la propietat mainwid.list la informació seleccionada pel botó i que està agafada del primer element de la subllista de botons.

  Un cop s'ha guardat la informació que correspon al botó seleccionat, aquesta funció també fa avançar la interfície a la següent pantalla. També s'encarrega d'esborrar tots els elements de la llista de botons, de manera que quan es torni a executar la mateixa pantalla es tornin a crear els botons i no apareguin duplicats.

7.2.4. Pantalla d'escaneig

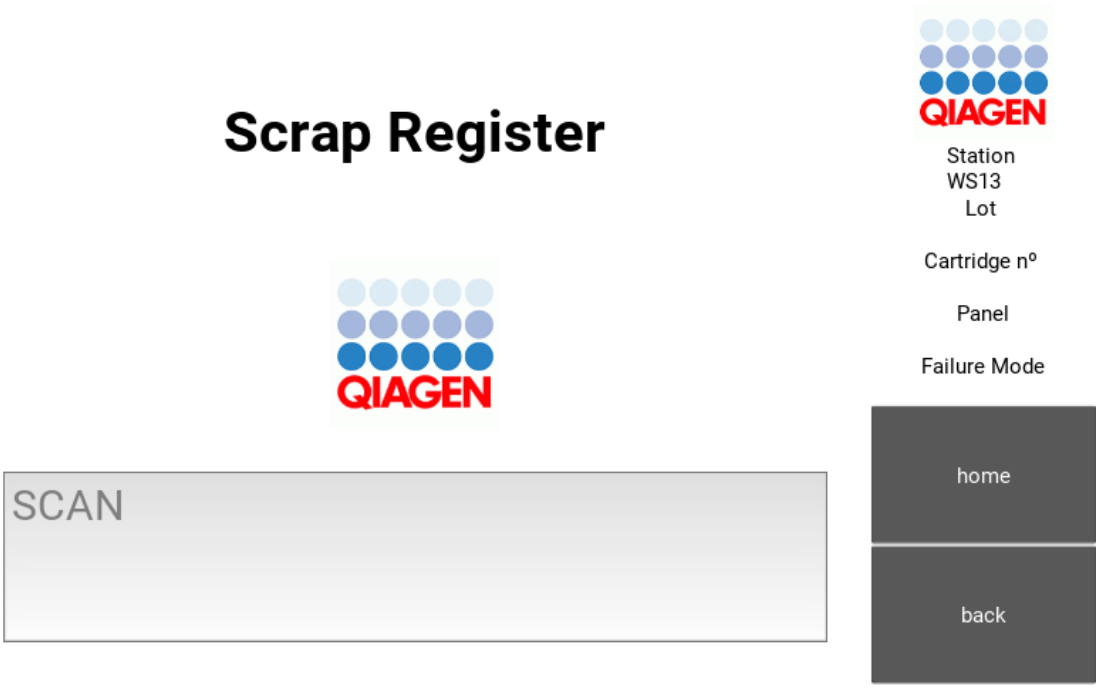
La tercera pantalla és diferent a totes les demés pel que fa la part de registre, si que comparteix la columna de la dreta amb la informació del registre actual, però la part que s'encarrega de registrar l'escaneig del cartutx, té una diferencia principal amb les demés pantalles. La part encarregada del escaneig és el widget ScanWid.

ScanWid és també un *Grid Layout*, però aquest ha estat dissenyat des de l'arxiu de disseny kv, ja que les eines d'escaneig no depenen de la informació emmagatzemada a la base de dades, i per tant no ha de ser creada cada vegada, així que ha sigut més fàcil de dissenyar. Aquesta compta amb tres widgets, una etiqueta amb el nom de la pantalla, una imatge amb el logotip de QIAGEN i un *TextInput*, amb el nom de "SCAN" que és la casella de text que s'ha de pitjar i seguidament fer l'escaneig del *DataMatrix*, per que la informació d'aquest pugui ser guardada per poder obtenir la informació del cartutx.

Aquesta informació es troba escrita a l'etiqueta, però aquesta es posa a la WS 15, per tant no sempre se sap tota aquesta informació, només es coniex el número de cartutx que surt junt a la *SN label* i el corresponent *DataMatrix*. Aquesta informació però també es troba en un altre format, en dos *DataMatrix* diferents, un senzill que és la SN *label*, que

es posa en primera instància al començar a fer el cartutx, i que per tant sempre hi serà, i l'altre és el que es posa més tard a l'estació 15, a la *Cartridge Label*, on hi ha tota la informació escrita, però també codificada en un *DataMatrix* més complex que l'anterior. Per tant, per tal d'estalviar temps escrivint a mà, i també pel fet de desconèixer la informació completa del cartutx, s'han d'escanejar un dels dos *DataMatrix* del cartutx amb una pistola escàner 2D.

En aquest cas, l'única complexitat que té aquesta part del programa és la funció scan(input), que és l'encarregada de fer tota la feina en aquest ScanWid.



Il·lustració 12 - Tercera Pantalla

7.2.4.1. Funcions

Com s'acaba de comentar l'única funció que té ScanWid és scan(input), on l'entrada és la informació que es registra amb la pistola de codis 2D, i la informació que escriu és una cadena de strings de 23 elements en el cas de SN Label, i de 269 elements en el cas del *Cartridge Label*.

El procés d'escaneig, s'encarrega d'enregistrar tres informacions pròpies del cartutx , nº de sèrie, lot i producte, els dos primers surten del SN que apareix enmig de la cadena de caràcters de les dues etiquetes, i el producte surt a partir d'un altre codi que és el GTIN que és el que identifica de cada producte que es fabrica a Stat-Dx. Per tant el primer que fa la funció és crear dues variables una per guardar el SN, i l'altre pel GTIN. A més a més

també s'aprofita en aquesta funció per enregistrar la data i la hora del escaneig gracies a la llibreria *datetime*..

Seguidament la funció ha de saber quin codi s'ha escanejat, i això es fa amb condicionals, sobre si té 23 elements, 269, o una quantitat diferent. En els dos primers casos es guarda en les corresponents variables el SN i el GTIN. L'últim cas, que pot ser un error a l'hora de fer el registre, i s'introdueixi una cadena de strings que no tingui cap de les dues llargades anteriors, salta un *popup*, el qual conté un botó que avisa al usuari de que el codi escanejat no és correcte, i que per tant s'ha de tornar a escanejar; quan es pitja el botó es torna a la pantalla d'escaneig i es torna a fer el registre del cartutx.

Quan la funció té la informació pròpia del cartutx, només queda obtenir la resta d'informació a enregistrar, que és la data i l'hora. Aquestes són molt fàcils d'obtenir amb la llibreria de python *datetime*, la qual té els mètodes datetime.today().year que diu l'any en el que s'ha cridat la funció, i el mètode datetime.now() que mostra la data i la hora; si es converteix en string i es fan sub strings, es pot guardar la informació del dia i la hora.

Un cop es té tot la informació a registrar, es guarda a la llista mainwid.cart, i ja està guardada tota la informació del cartutx excepte el MFM.

7.2.5. Pantalla de confirmació

És l'ultima part del programa i té l'única finalitat de fer un resum del registre que s'ha fet i acceptar-ho o rebutjar-ho en funció de si s'ha fet com calia, a criteri de l'operari.

Aquesta última pantalla és un *BoxLayout* i es diu ConfirmWid, és una còpia del ja mencionat InfoWid, ja que mostra la mateixa informació, l'única diferencia és la botonera que en aquest cas en comptes de navegar, permet acceptar o rebutjar el registre.

Consta de la primera imatge amb el logotip de l'empresa QIAGEN, 5 etiquetes que mostren la informació registrada (WS, SN, Lot, Producte i MFM) i finalment un últim calaix amb dos botons per acceptar o rebutjar. No es tornaran a explicar ja que funcionen igual que els descrits al punt 8.2.2, el que sí que canviarà seran les funcions que en aquest cas hi haurà una que s'encarregui d'acceptar el registre i l'altre de rebutjar-ho.

Il·lustració 13 - Cinquena Pantalla

7.2.5.1. Funcions

Les funcions d'aquest ConfirmWid són les que s'assignen als botons d'acceptar i rebutjar el registre. També hi ha la funció per actualitzar la informació quan es vagi a la pantalla de confirmació, així que updateinfo() també està inclosa dins la classe ConfirmWid.

- confirm(): És la encarregada d'acceptar el registre, i que per tant tanca el cicle del programa. El primer que fa es registrar el *scrap* a la base de dades amb la funció registercart(mainwid.cart), que registra la informació guardada a la llista mainwid.cart, immediatament després torna a buidar aquesta llista per tal que el proper registre no tingui restes del anterior escaneig. Per aclarir-li a l'operari de que el registre ha estat efectuat correctament, salta un *PopUp* que li dona el missatge de que el cartutx ha estat enregistrat correctament. Després d'això, el programa porta de nou a la pantalla de registre de WS per tal d'estar preparat pel proper registre.

- reject(): És molt més simple, l'únic que fa és esborrar tota la informació guardada a la llista mainwid.cart i torna a la primera pantalla de registre de DS, per tal de tornar a fer el registre.

7.3. Gestió de dades

La gestió de dades és una part molt important també del funcionament del sistema de

registre, ja que com s'ha vist, una de les principals característiques d'aquest és que agafi la informació que representa a l'app d'una base de dades i alhora la guardi registrada en aquesta mateixa.

Per una part s'explicarà com és la base de dades utilitzada, i per separat s'explicarà el fitxer database.py que és un fitxer de python que conté totes les funcions que interaccionen amb la base de dades.

7.3.1. Base de dades mfm

La base de dades utilitzada ha estat en un servidor de MariaDB, i és una base de dades que té com a objectiu tenir la informació de totes les estacions de registre (DS), de totes les estacions de treball (WS), dels productes amb tota la informació de cada producte (GTIN) i també tots els possibles errors i motius de rebuig (MFM), aquesta informació està guardada en un taula diferent per a cada una, i a més a més n'hi ha una altre que es diu *manufacturing* que és on es faran tots els registres. Aquestes taules han estat proporcionades per l'empresa i han sigut modificades per poder tenir tots els camps necessaris per al registre, però també té alguns camps que l'empresa ha volgut que estiguin tot i que no hagin estat utilitzats.

La base de dades utilitzada es troba en un servidor i el programa sempre intentarà treballar amb aquesta, però en el cas de pèrdua de connexió el programa ha d'estar preparat per ser capaç de seguir funcionant, així que la base de dades mfm del servidor també es trobarà de forma local en el dispositiu de registre, de manera que el programa pugui consultar aquesta base de dades local en cas de pèrdua de connexió. A més a més, el programa compta amb un arxiu de seguretat amb els últims registres fets aquest cada un cert temps mira si la base de dades ha enregistrat o no els últims cartutxos i si no ho ha fet puja la informació que ha estat guardada en l'arxiu de seguretat.

7.3.1.1. Taula discardstation

Aquesta primera taula, és la que conté la informació relacionada amb les estacions de rebuig, les DS, i és molt simple ja que està formada simplement per 8 entrades, una per cada DS que hi ha definida a la línia de producció. La taula conté simplement quatre camps i són els següents:

- Id: El primer i serveix per poder tenir identificades les entrades i poder recórrer la taula, aquest id és de tipus Int, i és un enter que es va incrementant d'un en un per cada entrada que es fa.

- Ds: És el segon camp taula i és del tipus Varchar, en aquest camp es col·loca el codi de totes les DS. Aquest codi és veurà a la botonera junt amb la descripció.

- Ws: Aquest camp també de Varchar serveix per tenir relacionada cada DS amb la WS que és l'última d'aquesta DS a la que pertany, això serveix per poder identificar la DS tant amb el seu propi codi com amb el de l'última WS d'aquesta, que recordant, és la WS on hi ha algun control de qualitat i es detecten els errors.

- *Description*: En aquest camp també del tipus Varchar hi ha la descripció que apareixerà més tard a l'app, i que diu quines WS pertanyen a cada DS, és una informació que està orientada a l'operari, de manera que pugui saber fàcilment quina DS escollir.

id	ds	ws	description
1	DS01	WS02	WS from 01 to 02
2	DS02	WS07	WS from 03 to 07
3	DS03	WS08	WS 08
4	DS04	WS09	WS 09
5	DS05	WS13	WS 10 to 13
6	DS06	WS16	WS 14 to 16
7	DS07	WS17	WS 17
8	DS08	WS18	WS 18

Taula 3 - Taula discardstation de la base de dades mfm

7.3.1.2. Taula Workstation

Aquesta segona taula és molt semblant a l'anterior, té 18 entrades, una per cada WS que hi ha a la línia de producció. La taula té cinc camps dels quals només quatre són rellevants i són els següents:

- Id: Igual que abans és el primer i serveix per poder tenir identificades les entrades i poder recórrer la taula, aquest id és de tipus Int, i és un enter que es va incrementant d'un en un per cada entrada que es fa.

- Ws: És un camp de tipus Varchar i en aquest es llisten els 18 codis de casa WS, aquest codi és el que després es veurà a la botonera junt amb la descripció de la WS.

- *Description*: Del tipus Varchar, serveix per donar a l'operari una descripció de la WS més tècnica, on s'explica quin procés es fa en aquella WS, aquesta informació és la que es veurà a l'app i amb la que l'operari podrà identificar més fàcilment quina WS vol seleccionar.

- Cost: un camp que no s'ha utilitzat però està posat per demanda de l'empresa, ja que es vol treballar amb el cost del *scrap* des de la base de dades en un futur.

- Ds: del tipus Varchar, simplement relaciona cada WS amb la DS a la que pertany.

id	ws	description	cost	ds
1	WS01	FF Sealing & Cartridge Labelling	0	DS01
2	WS02	Filters & Frits assembly & FF CV	0	DS01
3	WS03	TC body &Cover Ultrasound Welding	0	DS02
4	WS04	TC lubrication & Insertion	0	DS02
5	WS05	Membranes & Purification Chamber Assembly	0	DS02
6	WS06	Purification Film Sealing	0	DS02
7	WS07	Filters Assembly & PF CV	0	DS02
8	WS08	BB Assembly & Insertion	0	DS03
9	WS09	PPMixes, PK & IC Dispensing	0	DS04
10	WS10	PPMixes, PK & IC Drying	0	DS05
11	WS11	RC Film Sealing	0	DS05
12	WS12	Reservoirs Film Sealing	0	DS05
13	WS13	RC, Reservoirs CV	0	DS05
14	WS14	DCC Assembly $ Cover Assembly	0	DS06
15	WS15	Cover Assembly & Labelling	0	DS06
16	WS16	IPC Leak Test & BB Filling	0	DS06
17	WS17	Gravimetric IPC 1 & Buffer Filling & Under reservoir Film Sealing & Gravimetric IPC 2	0	DS07
18	WS18	Cartridge packaging & Gravimetric IPC 3	0	DS08

Taula 4 - Taula workstation de la base de dades mfm

7.3.1.3. Taula products

Taula que s'encarrega de relacionar el GTIN característic de cada producte que es fabrica a l'empresa (que està contingut dins del *DataMatrix* escanejat) amb la descripció del producte de manera que es pugui mostrar a l'app de quin producte es tracta. Aquesta taula està formada per 5 camps diferents.

- Id: Com totes les altres taules hi és, i serveix per poder tenir identificades les entrades i poder recórrer la taula, aquest id és de tipus Int, i és un enter que es va incrementant d'un en un per cada entrada que es fa.

- GTIN: Del tipus Varchar i és el codi identificador de producte, és el que es troba contingut dins la informació escanejada, i serveix per comparar el GTIN registrat amb el que es troba en aquest camp per tal d'identificar el producte.

- *Product*: aquest camp mostra el codi intern de producte, simplement serveix per tenir més informació sobre el producte, és del tipus Varchar.

- *Name*: camp Varchar, conté el nom abreviat del producte i és el que sortirà representat al calaix d'informació de l'app.

- *Description*: Dóna una descripció més acurada del producte, serveix per donar més informació a qui consulti la taula de la base de dades, aquest camp és del tipus Varchar.

id	GTIN	Product	name	description
1	8437015450018	110001	\N	DiagCORE Analytical Module
2	8437015450025	110002	\N	DiagCORE Operational Module
3	8437015450124	n/a	\N	For R&D Use
4	18437015450121		\N	For R&D Use
5	8437015450407	210902	Empty	Empty Cartridge
6	18437015450404		Empty	Empty Cartridge
7	8437015450414	210901	\N	Fluorophore Cartridge
8	18437015450411		\N	Fluorophore Cartridge
9	8437015450421	210903	ACC	Analyzer Check Cartridge
10	18437015450428		ACC	Analyzer Check Cartridge
11	8437015450148	210002	DiagC Resp CE	DiagCORE Respiratory Panel 2 (CE Version)
12	18437015450145		DiagC Resp CE	DiagCORE Respiratory Panel 2 (CE Version)
13	8437015450964	210002	DiagC Resp CE	DiagCORE Respiratory Panel 2 (IUO and R&D Purposes
14	18437015450961		DiagC Resp CE	DiagCORE Respiratory Panel 2 (IUO and R&D Purposes
15	8437015450155	210004	DiagC Gastr CE	DiagCORE Gastrointestinal Panel 2(CE Version)
16	18437015450152		DiagC Gastr CE	DiagCORE Gastrointestinal Panel 2(CE Version)
17	8437015450162	210004	DiagC Gastr CE	DiagCORE Gastrointestinal 2 Panel (IUO and R&D Pur
18	18437015450169		\N	DiagCORE Gastrointestinal 2 Panel (IUO and R&D Pur
19	4053228033905	1114556	QI Resp CE	QIAstat-Dx® Respiratory Panel CE-IVD
20	14053228033902	691211	QI Resp CE	QIAstat-Dx® Respiratory Panel CE-IVD
21	4053228034643	210031	QI Resp IUO	QIAstat-Dx® Respiratory Panel IUO
22	14053228034640	1116391	QI Resp IUO	QIAstat-Dx® Respiratory Panel IUO
23	4053228034216	1115291	QI Resp FDA	QIAstat-Dx® Respiratory Panel FDA
24	14053228034213	691221	QI Resp FDA	QIAstat-Dx® Respiratory Panel FDA
25	4053228033912	1114563	QI Gastr CE	QIAstat-Dx® Gastrointestinal Panel CE-IVD
26	14053228033919	691411	QI Gastr CE	QIAstat-Dx® Gastrointestinal Panel CE-IVD
27	4053228034636	1116422	QI Gastr IUO	QIAstat-Dx® Gastrointestinal Panel IUO
28	14053228034633	1116390	QI Gastr IUO	QIAstat-Dx® Gastrointestinal Panel IUO
29	4053228034667	1116961	DC4	DC4 cartridge
30	14053228034664		DC4	DC4 cartridge
31	4053228034650	1116425	QI Men IUO	QIAstat-Dx® Meningitis Panel IUO
32	14053228034657	1116402	QI Men IUO	QIAstat-Dx® Meningitis Panel IUO

Taula 5 - Taula products de la base de dades mfm

7.3.1.4. Taula mfm

És la taula que conté més informació, en aquesta hi ha emmagatzemats tots el MFM

enregistrats i definits a l'empresa per l'equip d'enginyeria que analitza el *scrap*, conté 78 entrades amb els 78 MFM registrats a l'empresa. Compta amb 5 camps dels quals només s'utilitzen 4, i són aquests:

- Id: Com totes les altres taules hi és, i serveix per poder tenir identificades les entrades i poder recórrer la taula, aquest id és de tipus Int, i és un enter que es va incrementant d'un en un per cada entrada que es fa.

- *Code*: és el codi que identifica quin MFM és, és del tipus Varchar i apareixerà representat a la botonera junt a la *description*.

- *Description*: Com s'acaba de dir apareixerà junt al codi a la botonera de la pantalla de selecció de MFM, en aquesta descripció s'explica breument de quin error es tracta, aquesta informació és vital per poder identificar l'error i registrar-ho ja que hi ha 78 possibles errors i és molt difícil memoritzar-los tots, per tant aquesta informació ajuda molt a l'operari, també és de tipus Varchar.

- Humerr: variable que nos s'ha utilitzat però ha estat demanada per l'empresa.

- Ws: camp amb dades de tipus Varchar, i és la que farà de pont entre la DS seleccionada i els MFM que s'han de mostrar, en aquest camp s'assigna a cada MFM l'última WS de cada DS d'on s'identifica aquest error. Amb aquesta relació es pot mostrar quins són els errors que poden trobar-se a cada DS.

id	code	description	humerr	ws
1	MFM001	Bad sealing of qPCR FLUIDIC NETWORK FILM	0	WS02
2	MFM002	Missing component: qPCR FLUIDIC NETWORK FILM	1	WS02
3	MFM003	Misalignment of qPCR FLUIDIC NETWORK FILM	0	WS02
4	MFM004	Double sealing of qPCR FLUIDIC NETWORK FILM	1	WS02
5	MFM005	qPCR FLUIDIC NETWORK FILM damaged	1	WS02
6	MFM006	Cartridge damaged during manufacturing	1	WS02
7	MFM007	Foreign particle in qPCR FLUIDIC NETWORK FILM	1	WS02
8	MFM008	Missing component: BB frit	1	WS02
9	MFM009	Frit damaged during insertion	1	WS02
10	MFM010	Frit improperly inserted	1	WS02
11	MFM011	Missing component: Filter	1	WS02
12	MFM012	Filter damaged during insertion	1	WS02
13	MFM013	Filter improperly inserted	1	WS02
14	MFM014	Missing component: Vent	1	WS02
15	MFM015	Vent damaged during insertion	1	WS02
16	MFM016	Vent improperly inserted	1	WS02
17	MFM017	Missing component: membrane	1	WS07
18	MFM018	Membrane improperly inserted	0	WS07

19	MFM019	Bad sealing of Purification FILM	0	WS07
20	MFM020	Missing component: Purification FILM	1	WS07
21	MFM021	Misalignment of Purification FILM	0	WS07
22	MFM022	Double sealing of Purification FILM	1	WS07
23	MFM023	Purification FILM damaged	1	WS07
24	MFM024	Foreign particle in Purification FILM	1	WS07
25	MFM025	Missing component: BB rotor	1	WS08
26	MFM026	Missing component: lipseal	1	WS08
27	MFM027	Lipseal improperly assembled	1	WS08
28	MFM028	PK not dispensed	0	WS09
29	MFM029	IC not dispensed	0	WS09
30	MFM030	PPMix not dispensed	0	WS09
31	MFM031	Less quantity of PK	0	WS09
32	MFM032	Less quantity of IC	0	WS09
33	MFM033	Less quantity of PPMix	0	WS09
34	MFM034	Double drop of PK	0	WS09
35	MFM035	Double drop of IC	0	WS09
36	MFM036	Double drop of PPMix	0	WS09
37	MFM037	Ppmix dispenser tray mispositioned	1	WS09
38	MFM038	Cartridge mispositioned in Ppmix dispenser	1	WS09
39	MFM039	Bad sealing of RC FILM	0	WS13
40	MFM040	Missing component: RC FILM	1	WS13
41	MFM041	Misalignment of RC FILM	0	WS13
42	MFM042	Double sealing of RC FILM	1	WS13
43	MFM043	RC FILM damaged	1	WS13
44	MFM044	Foreign particle in RC FILM	1	WS13
45	MFM045	Bad sealing of Reservoir FILM	0	WS13
46	MFM046	Missing component: Reservoir FILM	1	WS13
47	MFM047	Misalignment of Reservoir FILM	0	WS13
48	MFM048	Double sealing of Reservoir FILM	1	WS13
49	MFM049	Reservoir FILM damaged	1	WS13
50	MFM050	Pneumatic IPC FAIL	0	WS16
51	MFM051	Dry Cartridge Weight IPC FAIL	0	WS17
52	MFM052	Spill of buffer R1	0	WS17
53	MFM053	Spill of buffer R2	0	WS17
54	MFM054	Spill of buffer R3	0	WS17
55	MFM055	Spill of buffer R4	0	WS17
56	MFM056	Spill of buffer R5	0	WS17
57	MFM057	Spill of buffer R6	0	WS17
58	MFM058	Spill of buffer R7	0	WS17
59	MFM059	Spill of buffer R8	0	WS17
60	MFM061	Cartridge mispositioned in Buffer filling	1	WS17
61	MFM062	Bad sealing of Under reservoir FILM	0	WS17
62	MFM063	Misalignment of Under reservoir FILM	0	WS17

63	MFM064	Double sealing of Under reservoir FILM	1	WS17
64	MFM065	Under reservoir FILM damaged	1	WS17
65	MFM066	Filled Cartridge weight IPC FAIL	0	WS17
66	MFM067	Weight IPC in Station 3 FAIL	1	WS18
67	MFM068	Cartridge not weighed in Station 1	1	WS17
68	MFM069	Cartridge not weighed in Station 2	1	WS17
69	MFM070	Cartridge not weighed in Station 3	1	WS17
70	MFM071	Dry Cartridge Weight Over Threshold	0	WS17
71	MFM072	Dry Cartridge Weight Under Threshold	0	WS17
72	MFM073	Filled Cartridge Weight Over Threshold	0	WS17
73	MFM074	Filled Cartridge Weight Under Threshold	0	WS17
75	MFM075	Spill of Buffers	0	WS17
76	MFM076	Barcode reader error	0	WS18
77	MFM077	Bad dispensed PPMixes	0	WS09
78	MFM078	Double membrane	0	WS07

Taula 6 - Taula mfm de la base de dades mfm

7.3.1.5. Taula manufacturing

Aquesta taula té una finalitat diferent de les demés, ja que no és una taula de consulta, sinó que és la taula on es faran tots els registres, per tant no té un número determinat d'entrades, ja que aquestes s'aniran fent a mesura que s'utilitzi el sistema de registre. D'altra banda si que té molts camps degut a que aquest sistema enregistra bastanta informació, i ha d'anar en camps diferents, en total hi ha 15, dels quals 5 no s'utilitzen en aquest sistema de registre, i per tant no s'explicaran.

- Id: Tot i tenir una finalitat diferent a les altres taules també hi és, i serveix per poder tenir identificades les entrades i poder recórrer la taula, aquest id és de tipus Int, i és un enter que es va incrementant d'un en un per cada entrada que es fa.

- Sn: de tipus Int, i guarda en nº de sèrie del cartutx obtingut en l'escaneig per tal de tenir traçabilitat dels cartutx escanejats.

- Lot: a partir del SN també es guarda el lot al qual pertany el cartutx enregistrat, serveix per tenir de forma més visual aquesta informació, també és de tipus Int.

- *Panel*: Amb el GTIN que s'obté de l'escaneig amb la pistola 2D, es pot saber a quin producte correspon, així queda enregistrat en l'entrada quin tipus de producte era el cartutx en qüestió.

- *Released*, ipc, step i value: Són quatre dels camps que no s'utilitzen en aquest sistema de registre.

- Mfm: De tipus Varchar, i guarda el codi del MFM seleccionat durant el registre.
- *Description*: És la breu descripció del MFM seleccionat, també de tipus Varchar.
- Ws: Guarda el codi de la WS seleccionada, de tipus Varchar.
- Cost: camp que ja apareixia a la taula de Workstation, com ja s'ha dit en aquest sistema de registre tampoc s'utilitza.
- Dp: Guarda el codi de la DS seleccionada, de tipus Varchar.
- *Date*: Per tenir traçabilitat i controlar com i quan s'ha fet el registre, es guarda en un camp de tipus Varchar quin dia ha estat efectuat.
- *Time*: A part de la data també es guarda per separat l'hora a la que s'ha efectuat el registre, això permet veure si un mateix cartutx s'ha escanejat més d'un cop i veure quin ha sigut l'últim i vàlid registre, també de tipus Varchar.

7.3.2. Arxiu de gestió de dades

Per separar l'aplicació de tot el que tenia a veure amb la base de dades, s'ha decidit deixar tot el que tingui interacció amb aquesta en un document apart que és database.py. En aquest hi ha totes les funcions que s'encarreguen de comunicar-se amb la base de dades d'alguna manera, ja sigui connectant-s'hi, consultant o enregistrant.

Per treballar amb bases de dades des de python s'ha utilitzat la llibreria mysql, que conté els mètodes necessaris per comunicar el programa amb la base de dades.

7.3.2.1. Funcions de connexió

Per a la connexió amb la base de dades, s'han creat dues variables, dos diccionaris amb la informació de *user*, *password*, *host*, *database* i la propietat raise_on_warnings (que sempre estarà en True per permetre que, si hi ha errors, puguin ser notificats) amb el nom de config.

Per veure si les bases de dades tenen connexió s'ha creat la funció checkcnx(), que s'encarrega de provar si hi ha connexió amb la funció *try*, i utilitzant el mètode mysql.connector.connect(**config), on la base de dades comprova si hi ha connexió, si n'hi ha retorna True. Amb la funció *exept* es comproven dos errors comuns en base de dades un per veure si hi ha problemes amb les credencials, i l'altre si no troba la base de dades mencionada en el servidor, altrament error, que no és conegut, en aquests casos fa un *print* sobre l'error, i retorna False.

7.3.2.2. Funcions de consulta

L'aplicació del sistema de registre mostra la informació que es troba a la base de dades, i per poder utilitzar aquesta informació s'han de poder fer consultes i obtenir la informació en un format que pugui ser utilitzat pel programa de python. Totes tenen en comú que el primer que fan, és comprovar que la base de dades té connexió abans de fer el registre, si en té ho fa a partir de la base de dades del servidor, si no ho fa a partir de la base de dades local. Les funcions de consulta són les següents:

- alldp(): Primer es crea la variable dp, que és el diccionari que utilitza el programa main.py per construir la interfície gràfica del programa. Després el que fa la funció és crear un cursor a la taula discardstation, i s'utilitza el mètode fetchall() del cursor que recorre totes les entrades de la taula i tracta a cada element recorregut com una llista amb tantes caselles com camps tingui aquest, llavors per a cada entrada es crea una clau al diccionari dp, amb el codi de DS i la descripció corresponent.

- allws(ds): Primer es crea la variable ws, que és el diccionari que utilitza el programa main.py per poder construir la botonera de la interfície gràfica del programa. Després la funció crea un cursor a la taula workstation, i s'utilitza el mètode fetchall() del cursor que recorre totes les entrades de la taula i tracta a cada element recorregut com una llista de la mateixa manera que abans; llavors per a cada entrada es comprova que la WS pertanyi a la DS que s'havia seleccionat anteriorment, per fer-ho simplement es compara la variable d'entrada ds de la funció amb el camp corresponent a la DS de la taula, i totes les entrades que pertanyin a la DS seran registrades al diccionari ws amb el codi de WS i la descripció corresponent.

- allmfm(ws): Primer se li assigna a la variable d'entrada un nou valor per que sigui compatible amb la informació de la base de dades, seguidament es crea la variable mfm, que és el diccionari que utilitza el programa main.py tenir la informació dels MFM que ha de representar. Després el que fa la funció és crear un cursor a la taula mfm, i s'utilitza el mètode fetchall() del cursor que recorre totes les entrades de la taula i fa el mateix que l'anterior funció, a mesura que recorre la taula compara el camp WS amb la variable d'entrada ws, si coincideix guarda aquesta entrada en el diccionari amb el codi de MFM com a clau i de valor la descripció.

- checkcartridge():Funció que té la intenció de retornar un booleà per saber si el registre que es vol fer ja ha estat fet o no, amb els camps de SN, hora i data iguals, cosa que ja fa impossible , que dos registres es facin al mateix temps i del mateix cartutx, això voldria dir que estan repetits.

Per donar aquesta informació, la funció el que fa, és agafar la informació de la llista cart que té com a variable d'entrada, després es crea un cursor a la taula manufacturing, i s'utilitza el mètode fetchall() del cursor que recorre totes les entrades de la taula per comprovar si hi ha alguna entrada que tingui el mateix valor en els camps de Sn, date i time, si és així la funció retorna True, indicant que el cartutx que es vol registrar ja ho ha estat; altrament si no troba cap entrada que compleixi aquesta similitud retorna False, volent dir que el cartutx no ha estat registrat encara.

7.3.2.3. Funcions de registre

Per últim queden les funcions de registre que són les indispensables per tal que la informació escollida durant el registre del *scrap* quedi guarda a la base de dades. Hi ha dues funcions, una s'encarrega de fer el registre de informació a la base de dades o al backup (en el cas que no sigui possible la connexió), l'altre s'encarrega de fer el registre a la base de dades des del fitxer de backup.

- registercart(cart): Funció encarregada de registrar la informació de la variable mainwid.cart del arxiu main.py a la base de dades, per tant la variable d'entrada d'aquesta funció, serà aquesta llista.

  Primer de tot es desglossa la llista en variables, una per a cada casella de la llista, llavors en funció de si hi ha connexió o no, el registre es fa a l'arxiu de backup.txt, o es fa a la base de dades. En el cas del *backup* simplement s'introdueix a l'arxiu una cadena de strings amb tota la informació separada per espais i un salt de línia al final. D'altra banda si hi ha connexió, primer de tot comprova que no hi hagi una entrada idèntica que la que es vol posar a la taula de manufacturing; després la funció crea una consulta INSERT a la taula de manufacturing i introdueix totes les dades en els camps corresponents de cop; amb la funció de cursor.execute que té dos variables, una que és la *query* de SQL que s'executarà, i l'altre les variables que s'introduiran en aquesta.

- registerbackup(): Aquesta funció fa el mateix que l'anterior, però en comptes d'agafar la informació de la llista, l'agafa de l'arxiu *backup*. Per llegir la informació del fitxer backup.txt, s'utilitza el mètode readlines(), que retorna una llista amb totes les línies que té el fitxer. Amb aquesta llista es fa un rsplit(' ') de la cadena de strings, que separa tots els elements que havien estat separats per un espai en blanc, i es creen les variables una a una de la mateixa manera que abans; després es segueix el mateix procediment i es fa l'entrada a la base de dades en el cas que no estigués feta abans. Aquesta funció s'executa repetidament cada 15 minuts, i si hi ha connexió i el cartutx encara no havia estat registrat ho fa.

7.4. Hardware

El sistema de registre necessita poder ser utilitzat per un operari a la línia de producció, per tant ha d'estar implementat amb un hardware, tant un computador que permeti treballar amb python i base de dades, com també els perifèrics indispensables, una pantalla per interaccionar, i un escàner de codis 2D per escanejar els cartutxos.

7.4.1. PC

L'ordinador és el component del sistema de registre que s'encarrega d'executar els programes que fan funcionar l'app i fer tots els càlculs i operacions necessàries durant l'execució. Cal un ordinador amb les següents característiques per desenvolupar totes les funcions:

- Rendiment mitja-baix per tenir prou potència per fer córrer el programa.
- Que utilitzi un sistema operatiu compatible amb python i Kivy.
- Que pugui treballar amb bases de dades, en concret MariaDB.
- Que sigui compacte i que no ocupi massa espai a la línia de producció.
- Que tingui connectivitat necessària, connexió Ethernet i port USB per a un lector de codis 2D.
- Que tingui suficient voltatge per subministrar corrent a perifèrics.

7.4.2. Perifèrics

Per donar suport al computador i poder veure i interaccionar amb aquest es necessiten perifèrics, per tal de poder utilitzar la interfície gràfica programada sense problema. Per tant els perifèrics han de complir les necessitats del programa i també les del projecte, que són els següents:

- Pantalla mida petita però suficient per poder veure correctament la interfície gràfica del programa.
- Sistema de control de la pantalla per seleccionar el panell de l'app.
- Escàner capaç de llegir codis 2D, QR, *DataMatrix*.
- Sistema de perifèrics reduït al màxim per evitar ocupar més espai del necessari.
- Compatibilitat amb el ordinador i el seu sistema operatiu.

8. Prototip

Després d'haver fet el disseny del sistema complet de registre de *scrap*, s'ha decidit muntar un primer prototip per veure com pot adaptar-se i instal·lar-se aquest dins de la línia de producció.

La intenció de la construcció del primer prototip és veure com s'implementa de forma econòmica aquest sistema de registre a la línia, i comprovar amb els operaris i amb els enginyers si podria millorar el sistema actual de registre manual, i l'estudi que fan els enginyers a partir d'aquest.

Per dur-ho a terme s'han comprat els components que s'han ajustat més a la idea d'aquest prototip, i s'han configurat i posat a punt per que puguin ser utilitzats per fer córrer el programa i així veure funcionar el sistema de registre de *scrap*.

8.1. Components

El prototip ha estat construït utilitzant només tres components per tal de minimitzar el número de perifèrics, ja que impliquen més connexions, més elements físics en el sistema, i per tant més espai, cosa de la qual no se'n disposa dins la línia de producció, s'ha escollit un miniordinador per fer de PC i gestionar totes les dades, una pantalla tàctil com a sistema de control i com a pantalla, i finalment una pistola escàner de codis 2D.

8.1.1. PC, Raspberry Pi 4 Model B

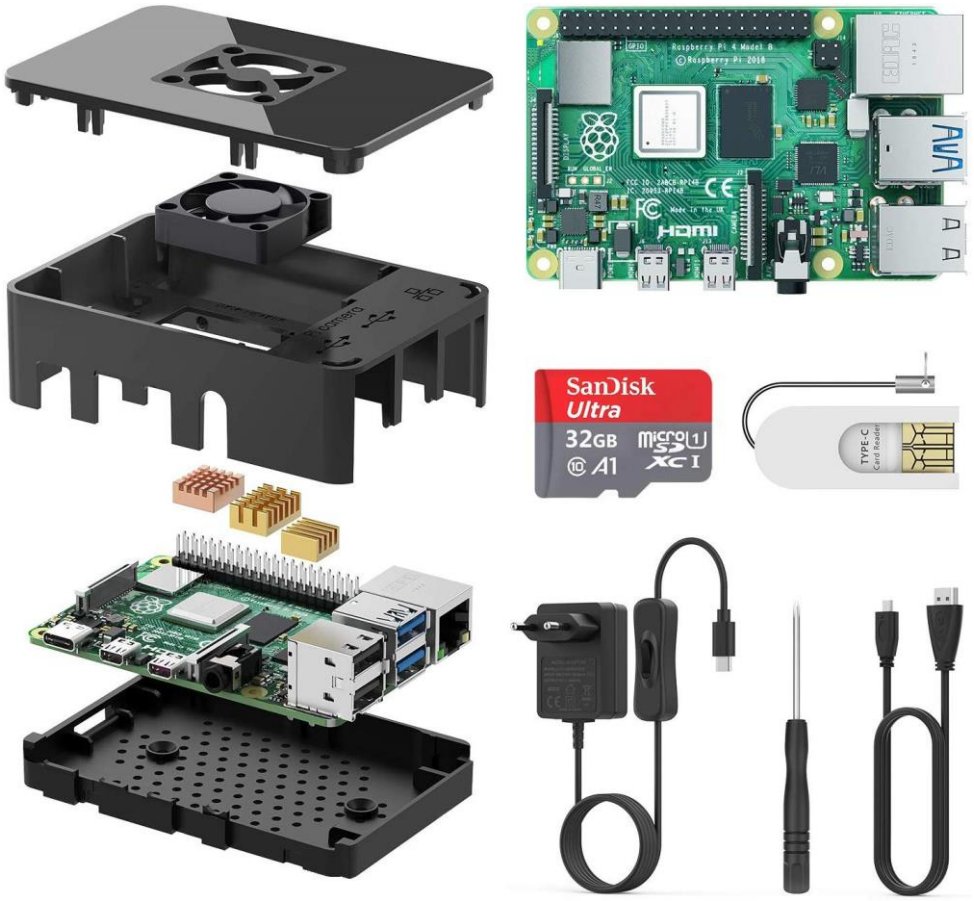
El component escollit per fer el prototip del sistema de disseny, ha estat un miniordinador de la marca Raspberry, una empresa que s'enfoca en dissenyar i distribuir els seus miniordinadors que tenen unes característiques molt bones, estan pensades per donar-li varietat d'usos des de la programació, a la domòtica, entre altres. La qüestió és que, és un ordinador que té un mida reduida, i una molt bona potència, a més a més, funciona amb Raspbian (per defecte), que és compatible amb Python ja que aquest SO està basat en Linux, i també porta preinstal·lat el Python.

El model escollit per al muntatge del prototip, ha estat el més potent per evitar problemes, i s'ha agafat un paquet de l'empresa TICTID, que proporcionava la Raspberry juntament amb altres accessoris que faciliten el muntatge, que són els següents:

- Raspberry:
    - Connectivitat: Bluetooth 5.0, WIFI dual (2.4 i 5.0 GHz) i Port Ethernet.
    - Entrades, 2 USB 2.0, 2 USB 3.0.

- Alimentació: USB Type-C amb 5V i 3A.
- Imatge: dos ports micro HDMI, fins resolució 4k.
- Emmagatzematge: port microSD fins a 64 Gb de capacitat.

- Caixa per acoblar la Raspberry i ventilador per dissipació.
- Dissipadors metàl·lics per a components.
- Cable d'alimentació amb interruptor.
- Targeta SD amb Raspbian instal·lat, amb adaptador USB.
- Cable Micro HDMI.
- Eines pel muntatge.



Il·lustració 14 - Kit muntatge Raspberry Pi 4 [19]

8.1.2. Pantalla tàctil Longruner

S'ha decidit que per a la implementació del sistema de registre, per complir els

requeriments, la millor opció era una pantalla tàctil, ja que permet evitar l'ús de ratolí i teclat (ja que si fes falta pot integrar-se un teclat en el programa). La mida de la pantalla ha de ser prou gran per poder llegir correctament la informació i també ser prou precís amb el control tàctil, però també prou petita per no ocupar massa espai, per això s'ha decidit escollir el model econòmic Longruner 7" LCD Display, que té les següents característiques:

- Pantalla: 7 polzades, LCD 1024x600.
- Imatge: 1 X entrada HDMI.
- Connectivitat: 1 X Micro USB per al Touch control.
- Alimentació: 1 X Micro USB per alimentació 5V (pot alimentar-se a partir de la Raspberry).
- Compatible amb Raspberry.

Il·lustració 15 - Pantalla tàctil Longruner [26]

8.1.3. Lector codis 2D, inateck

Pel cas de la pistola encarregada de llegir els codis 2D, concretament *DataMatrix*, s'ha escollit aquesta pistola, ja que reconeix els dos tipus de codi que són necessaris d'escanejar per al registre, entre altres; d'altra banda compta amb múltiples configuracions d'escaneig que poden ser útils per al projecte. En aquest cas s'ha buscat una pistola de gamma mitjana, ja que per a un registre eficaç es necessita una bona eina

d'escaneig, ja que errors durant l'escaneig poden provocar una pèrdua de temps innecessària. Tampoc però, s'ha escollit la millor opció ja que el preu es dispara. Les característiques de la pistola són les següents:

- Alimentació: 5V.
- Connectivitat: USB.
- Codis 2D: QR-CODE, DATA MATRIX, PDF417, Micro PDF417, Micro QR Code, Aztec, MaxiCode, Rectangular Data Matrix
- Compatibilitat amb Raspberry Pi.
- Accessori inclòs, Holder.

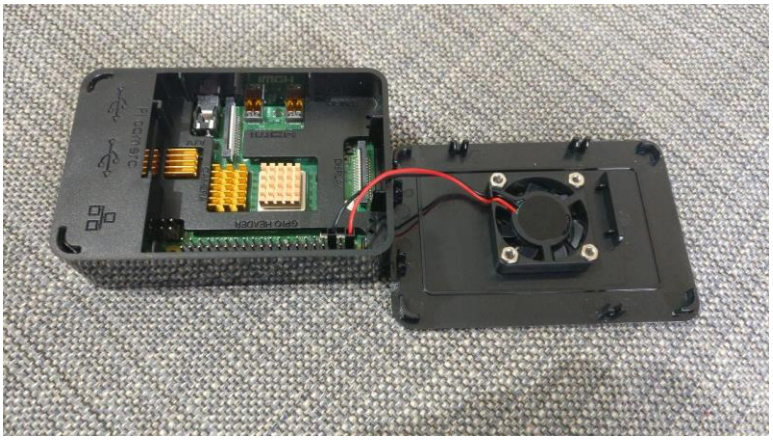
Il·lustració 16 - Pistola escàner 2D Inateck [20]

8.2. Muntatge i Configuració

El muntatge del prototip ha sigut molt senzill, ja que simplement es tracta d'unir components i poc més, de totes maneres els passos seguits han estat:

1. Muntar el kit de la Raspberry:

   Primer de tot cal enganxar els 3 dissipadors sobre els tres components indicats, que són el CPU, GPU i RAM, després es col·loca la placa sobre la part inferior de la caixa. Per Muntar la tapa de la caixa, es colla el ventilador amb els quatre cargols. Finalment s'ajunten les dues parts de la caixa, i es connecta el ventilador en els pins de la placa que convinguin, segons es vulgui més o menys potència. Per que la Raspberry funcioni, només cal endollar l'alimentació en el port tipus C i introduir la targeta SD a la part inferior de la placa, amb això el kit ja està muntat.



Il·lustració 17 - Raspberry muntada dins la carcassa

2. Configurar la Raspberry:

   L'ordinador ja és funcional, però abans de connectar-ho amb la pantalla i l'escàner, cal posar-ho a punt per poder córrer el programa sense problemes.

   Primer de tot s'ha hagut de tornar a instal·lar python, ja que la versió instal·lada de fàbrica era la 2.8, i no és compatible amb el programa desenvolupat, ja que ha estat fet amb python 3.7, així que aquesta és la versió que s'ha instal·lat i posat com a predeterminada.

   També s'ha hagut d'instal·lar les llibreries de Kivy i les de SQL, ja que no venen amb python. Kivy per reconèixer les interaccions amb la pantalla, s'ha de configurar, ja que de forma predeterminada només ve amb reconeixement per ratolí, i s'ha d'afegir el control tàctil de la pantalla.

   L'última part a configurar són les bases de dades, primer de tot cal instal·lar MariaDB a l'ordinador per que pugui manegar les bases de dades, i seguidament importar la base de dades més actualitzada amb tota la informació, per tal que el sistema pugui funcionar sense accés al servidor.

   Quan ja té tota la configuració necessària, només cal importar el programa de

python que fa el procés de registre juntament amb els altres documents complementaris, i amb això ja pot funcionar l'app a la Raspberry.

3. Connectar perifèrics

   Un cop la Raspberry està llesta per funcionar, s'ajunten tots els components. L'ordinador és l'element central, des del que s'alimenten els dos perifèrics, per tant cal començar endollant la Raspberry. Per connectar els perifèrics, s'utilitza un USB per la transmissió d'informació i alimentació de la pistola escàner; d'altra banda s'utilitza un USB per alimentar la pantalla, un altre USB per transmetre la informació de l'ús tàctil de la pantalla, i finalment un cable HDMI per transmetre la imatge.

4. Sistema muntat

   En aquest punt ja es pot utilitzar el sistema de registre encenent l'interruptor del cable d'alimentació de la Raspberry i també la pantalla.

Il·lustració 18 - Sistema de registre de scrap operatiu

8.3. Impacte ambiental

El projecte té un impacte ambiental directe pel que fa l'ús de materials per a la construcció del dispositiu, ja que aquests són components electrònics, a banda dels plàstics que els acompanyen, la part dolenta és que són un dels tipus de residus més difícils de reciclar, ja que estan formats per metalls i plaques de silici. Els metalls comuns i els plàstics i vidres tenen un percentatge de reciclatge alt, però els metalls més específics

com l'or o el sílice, són molt més difícils de reciclar. Tot i així el reciclatge electrònic cada vegada és més rendible, això fa que es millori l'eficiència d'aquest cada vegada més.

D'altra banda, aquest projecte permet fer un registre del rebuig de la línia de forma digital, utilitzant només la pantalla, l'ordinador i l'escàner, que són components que tenen una vida útil d'entre uns 5-10 anys, i només necessiten alimentar-se de la xarxa elèctrica amb una potència total de 15 W, un consum molt baix.

Aquest sistema digital substitueix a l'antic sistema de registre que es fa a mà, en el qual s'utilitzen fulles de paper i bolígraf, ometent el bolígraf que té un impacte mediambiental molt baix, si es compta la quantitat de paper diària que es gasta només per fer aquest registre, es pot arribar a xifres molt grans, ja que el paper és un consumible que s'utilitza a diari per fer al registre, a més a més és paper imprès cosa que fa que el reciclatge d'aquest sigui menys net.

En conclusió, la implementació d'aquest sistema de registre permet reduir la despesa de paper, que és un dels principis del reciclatge més importants (reduir el consum). Amb aquesta implementació es passarà de crear tones de paper que s'ha de llençar al llarg dels anys, a crear aproximadament un kilogram en total de deixalles electròniques, que poden reciclar-se.

9. Planificació i costos

En aquesta part es detalla quantes hores se li han dedicat al projecte, i quins han estat els costos per poder desenvolupar el sistema de registre de *scrap*, i més endavant adquirir el components per poder construir el primer prototip.

9.1. Planificació

Per poder explicar com s'han organitzat les tasques durant la realització del projecte, s'explicarà cada tasca i que ha comportat cada una.

La primera etapa del projecte ha estat l'estudi preliminar, on s'ha hagut d'identificar quin és el problema que hi havia a la fàbrica a l'hora de registrar el scrap. Dins d'aquesta primera etapa, també es contempla l'estudi de la línia de producció i l'organització d'aquesta en DS, i també classificant WS i MFM. Finalment a partir d'aquí s'han establert els objectius.

Abans de començar amb el disseny del programa, s'ha hagut de fer recerca sobre les possibilitats d'implementació que tenia el sistema que es volia dissenyar. També se n'ha fet sobre llenguatge python, i sobretot de la seva GUI Kivy; també calia aprendre a fer servir bases de dades y veure de quina manera es podia introduir en el projecte.

Amb les eines clares, ha començat la fase de disseny. Aquesta ha sigut l'etapa que ha necessitat una major inversió de temps, ja que s'ha hagut de crear el programa des de zero. D'altra banda també s'ha fet la base de dades per tal que funcionés correctament amb el programa.

Per últim com a culminació del treball, s'ha construït el prototip. Primer de tot s'ha fet una recerca sobre els components que podrien ser més adients, un cop comprats s'ha muntat de forma molt senzilla. Amb el prototip muntat el que ha sigut més difícil ha estat la configuració i la posada en marxa d'aquest, ja que a l'hora de fer-ho funcionar han anat aparegut diferents errors, que s'han hagut d'anar solucionant fins que el programa s'ha executat com era degut.

Etapa	Tasca	Hores
Estudi Preliminar	Identificació del problema	20
	Estudi de la línia de producció	40

	Definició d'objectius	10
Recerca	Recerca sobre python i Kivy	50
	Recerca SQL i MariaDB	20
Disseny	Disseny programa	160
	Disseny base de dades mfm	50
Prototip	Cerca de components	10
	Construcció del prototip	10
	Configuració prototip	40
Total		410

Taula 7 - Planificació d'hores

Comptant les hores que se li han dedicat al projecte, en total sumen 410. Durant aquesta estona,l'enginyer ha estat fent feina, ja sigui fent recerca sobre el projecte, estudiant algun aspecte o dissenyat i construint el sistema de registre de rebuig.

9.2. Costos

Per una part hi ha hagut els costos de disseny del sistema de registre, que té a veure amb les hores que ha hagut de pagar l'empresa a l'enginyer per desenvolupar aquest sistema de forma que sigui funcional.

L'enginyer ha trigat 4 mesos (16 setmanes) en desenvolupar el sistema de registre programant la app i la base de dades, amb una dedicació setmanal de 20 hores, tenint en compte que l'enginyer cobra a raó de 30 euros la hora.

A més a més l'enginyer també s'ha dedicat a muntar el prototip i posar-ho a punt per poder fer-ho servir, en aquesta fase de muntatge i posada a punt li ha dedicat 3 setmanes amb la mateixa dedicació de 20 hores setmanals.

D'altra banda també s'ha de tenir en compte el cost de les eines utilitzades per desenvolupar el projecte. En aquest cas al ser un projecte que es centra en el desenvolupament d'un programa informàtic, només ha sigut necessari un ordinador que ha proporcionat l'empresa, que costa aproximadament 1000 euros. Si es considera que la vida útil d'un ordinador es de 5 anys, que passant-ho a hores (comptant que cada dia

s'ha utilitzat 4 hores), són unes 7.300 hores. Per tant, l'ús de l'ordinador té un preu per cada hora d'ús de

D'altra banda pel que fa l'ús de software, no s'ha comptabilitat cap cost, ja que els programes utilitzats han estat tots de llicència lliure i per tant gratuïts.

També s'han de tenir en compte els costos del material que han sigut necessaris per al prototips, l'empresa ha decidit comprar material necessari per fer dos prototips d'entrada.

Concepte	Unitats	Hores	Preu hora/ unitat	Preu
Desenvolupament programa	-	350	30€	10.500 €
Muntatge i posada a punt del prototip	-	60	30€	1.800€
Ordinador	-	400	0,14€	54,8€
Kit Raspberry Pi 4 de TICTID	2	-	109,99€	219,98€
Pantalla tàctil Longruner	2	-	65,99€	131,98€
Pistola codis 2D Inateck	2	-	59,99€	119,98€
Total	-	-	-	12.826,74€

Taula 8 - Càlcul de costos totals

Tenint en compte els diners que ha costat el desenvolupament del sistema de registre de *scrap* i el muntatge del prototip, i li afegim el preu dels materials necessaris per al prototip, tot suma la quantitat de 12.826,74€.

Conclusions

El projecte ha englobat moltes tasques i diferents anàlisi, que han portat a complir l'objectiu principal d'aquest projecte que és desenvolupar un sistema de registre digital per al rebuig de línia o *scrap*, cada part ha requerit d'uns esforços i una dedicació diferents, per tan es passarà per cada un d'aquest punts.

Primer de tot, la necessitat de saber com està estructurada la línia de producció, com es distribueix en diferents estacions de treball, i quins errors es poden produir a cada una, ha permet aprofundir i aprendre sobre el producte que es fàbrica a l'empresa, entenent el procés de manufactura del cartutx des d'un altre punt de vista. D'altra banda, durant l'anàlisi del problema s'ha pogut concloure que el *scrap* té una importància molt gran pel que fa el producte, ja que aporta informació útil per a la millora de la fabricació i també a possibles millores de disseny del cartutx, no deixant de banda que dona una visió actual de com s'està fabricant a la línia de producció.

Durant el disseny del sistema de registre, és a dir l'app, s'ha hagut d'investigar sobre les diferents possibilitats que hi havia per implementar aquest sistema. Gràcies a les característiques que s'han imposat, s'ha descobert el Kivy, un mòdul de python per desenvolupar GUI, que ha resultat ser molt potent i permet fer tot tipus d'interfícies. També cal recordar que aquest sistema s'ha implementat utilitzant bases de dades, un sistema del que també s'ha hagut d'aprendre des de zero. La fase de desenvolupament de l'app, ha explotat la capacitat d'autoaprenentatge al màxim, ja que programant projectes diferents es creen necessitats diferents i per tant solucions diferents que utilitzen nous mètodes, l'autoaprenentatge ha estat un punt molt destacat del disseny del sistema de registre.

Finalment, el fet de que aquest projecte culmini amb la implementació d'aquest mitjançant un primer prototip, dóna una visió excel·lent del resultat obtingut després de la feina feta en aquest projecte. El prototip permet veure com s'utilitza el sistema de registre a la realitat i veure com desenvolupa la seva tasca.

Després d'haver dut a terme el projecte i veure els resultats obtinguts, amb l'app creada i el prototip muntat i funcionant, surten alguns aspectes que s'han observat que podrien millorar-se en un futur.

El programa que executa la Raspberry, és la primera versió, i compta amb les funcionalitats bàsiques per dur a terme la seva funció de registre, durant el desenvolupament del programa s'han detectat errors del programa que s'han corregit sobre la marxa, però hi ha alguns que segur que s'han passat per alt i no es detectaran fins que no se l'hi doni un ús més intensiu al sistema de registre.

Parlant del programa també poden sorgir noves millores d'aquest, com la part estètica, o també afegir la funcionalitat d'escriure un MFM nou, que escrigui l'operari.

D'altra banda pel que fa el prototip, s'ha detectat que la qualitat d'aquest no és excepcional, ja que està pensat per ser un prototip que permeti veure com s'implementa el sistema a la línia. Durant la prova del prototip, s'ha detectat que la Raspberry a vegades no és un dispositiu amb prou potència per fer anar el programa, i a vegades presenta un cert retard, per tant s'hauria de triar un PC amb més potència. Pel que fa la pantalla, és una mica petita al ser de 7" i resulta incòmode interaccionar amb el programa amb aquesta, per tant fa pensar que seria més convenient una utilitzar una pantalla de 10", per últim l'escàner dona un resultat molt bo, tot i que també es podria plantejar escollir un model de més qualitat per detectar qualsevol codi 2D en qualsevol situació de llum, posició, etc. A més a més tot el cablejat per unir els components queda al descobert cosa que pot arribar a generar problemes.

Per englobar aquestes característiques per un proper prototip, es planteja utilitzar un Panel PC. Un ordinador orientat a l'ús industrial que porta integrat una pantalla tàctil, tot junt, d'aquesta manera les connexions entre si queden tapades en un sol dispositiu. Existeix una gamma de panel PC amb diferents potències i pantalles, que es podrien adaptar perfectament a les necessitats. L'inconvenient d'aquest tipus d'ordinador és que són molt més cars que la Raspberry i la Pantalla Longruner, a més a més també són d'una mida més gran.

Agraïments

Aquest projecte ha estat desenvolupat a l'empresa Stat-Dx, on he realitzat la meva estada de pràctiques, per tant he d'agrair en primer lloc al meu tutor a l'empresa L'Àlex Boada per proposar-me el projecte que tenien pensat el cap d'enginyeria Yuhan Kim i el meu company L'Ignacio Micolau, que ha estat a sobre del treball en tot moment donant-me tot el recolzament possible per dur-ho a terme. També he d'agrair molt al Daniel Atienza, de l'equip d'enginyeria, que m'ha ajudat amb el tema de les bases de dades i com utilitzar-les en el meu projecte. Per últim agrair a totes les altres persones dels departaments de producció, qualitat i enginyeria que també han estat donant-me suport.

D'altra banda, des de l'escola, he d'agrair molt la feina del meu tutor Lluís Solano que m'ha ajudat a guiar i a mostrar-me com dur a terme aquest TFG, i donar-me el cop de mà que em calia quan ho necessitava.

10. Bibliografia

[1] APPLICATIONS FOR PYTHON [https://www.python.org/about/apps/ ; 10 d'octubre 2019]

[2] PYAR, INTERFÏCIES GRÀFIQUES [http://www.python.org.ar/wiki/InterfacesGraficas ; 29 d'octubre 2019]

[3] DEV, GUI PROGRAMING WITH PYTHON [https://dev.to/amigosmaker/gui-programming-with-python-spanish-2889 ; 1 de novembre 2019]

[4] KIVY DOCUMENTATION [https://kivy.org/doc/stable/api-kivy.html ; 10 de novembre 2019]

[5] W3 SCHOOLS, DOCUMENTACIÓ DE PYTHON [https://www.w3schools.com/python/ ; 10 de novembre 2019]

[6] LIKEGEEKS, TUTORIAL DE KIVY – CONSTRUYE APLICACIONES CON INTERFACES GRAFICAS USANDO PYTHON [https://likegeeks.com/es/tutorial-de-kivy/ ; 15 de novembre 2019]

[7] PYTHON WIKI, GUI PROGRAMING IN PYTHON [https://wiki.python.org/moin/GuiProgramming ; 15 de novembre 2019]

[8] GITHUB, DOCUMENTACIÓN i PRÁCTICAS KIVY IVANDRAGOGEAR [https://github.com/IvanDragogear/Practicas_kivy_espanol ; 20 de novembre 2019]

[9] KIVY CON PYTHON PARA EL DESARROLLO DE APLICACIONES MÓVILES [https://unipython.com/kivy-python-desarrollo-aplicaciones-moviles/ ; 20 de novembre 2019]

[10] MARIADB DOCUMENTATION [https://mariadb.org/documentation/ ; 25 de novembre 2020]

[11] SISTEMAS DE GESTION DE BASE DE DATOS | TIPOS Y CLASIFICACIÓN [https://www.tecnologias-informacion.com/gestionbasedatos.html ; 25 de novembre 2019]

[12] EL LENGUAJE SQL [https://www.mundoracle.com/el-lenguaje-sql.html?Pg=sql_plsql_2.htm ; 26 de novembre 2019]

[13] DOCS PYTHON, CLASES [http://docs.python.org.ar/tutorial/3/classes.html ; 30 de

novembre 2019]

[14] DOCS PYHTON, DATETIME [https://docs.python.org/3/library/datetime.html ;30 de novembre]

[15] CHAPTER 1 INTRODUCTION TO MYSQL CONNECTOR / PYTHON [https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlconnection-raise-on-warnings.html ; 1 de desembre 2019]

[16] INSTALACIÓN COMPLETA DE BASE DE DATOS MYSQL & MARIADB EN RASPBERRY PI 3 B7 B+ [http://pdacontroles.com/instalacion-completa-base-de-datos-mysql-mariadb-en-raspberry-pi-3-b-b/ ; 15 de gener 2020]

[17] INFAIMON, PC PANEL [https://www.infaimon.com/producto/pc-panel/ ; 20 de gener 2020]

[18] RASPBERRY PI 4 [https://www.raspberrypi.org/products/raspberry-pi-4-model-b/ ; 21 de gener 2020]

[19] FOTOGRAFIA KIT RASPBERRY, INFORMACIÓ DEL KIT [https://www.tictid.com/product/b07zf73rb5.html ; 25 gener 2020]

[20] BCST-51 2D USB WIRED BARCODE SCANNER, READ SCREEN [https://www.inateck.com/inateck-2d-usb-wired-code-scanner-read-screen-bcst-51.html ; 25 de gener 2020]

[21] QIAGEN HOME [https://QIAstat-Dx®.com/row/ ; 25 de febrer 2020]

[22] INTRODUCTION TO PCR PRIMER & PROBE CHEMISTRIES [https://www.bio-rad.com/en-ch/applications-technologies/introduction-pcr-primer-probe-chemistries?ID=LUSOJW3Q3 ; 22 de març 2020]

[23] BUFFERS [https://chem.libretexts.org/Bookshelves/Physical_and_Theoretical_Chemistry_Textbook_Maps/Supplemental_Modules_(Physical_and_Theoretical_Chemistry)/Acids_and_Bases/Buffers ; 22 de març 2020]

[24] CÓDIGOS BIDIMENSIONALES QR, BIDI Y DATAMATRIX: ¿CUÁL ELEGIR? [https://www.etiquetas-laboratorio.com/blog/codigos-bidimensionales-qr-bidi-datamatrix/ ; 25 de març 2020]

[25] ORACLE DATABASE ONLINE DOCUMENTATION 11G RELEASE 1 [https://docs.oracle.com/cd/B28359_01/nav/portal_4.htm ; 25 de març 2020]

[26] IMATGE, LONGRUNER FOR RASPBERRY PI TOUCH SCREEN 7 INCH 1024X600 LCD TFT DISPLAY HDMI MONITOR WITH PROTECTIVE CASE FOR RASPBERRY PI 3 2 1 MODEL B B+ A BB BLACK PC [https://lebanon.desertcart.com/products/3871129-the-ultimate-kit-with-arduino-uno-from-oddwires ; 27 de març 2020]

[27] BASURA ELECTRÓNICA, RECICLAJE Y ECONOMÍA CIRCULAR [https://ambientaldata.com/informacion/reciclaje-de-basura-electronica-y-economia-circular/ ; 1 d'abril 2020]

11. Annexos

11.1. Codi main.py

```
from kivy.config import Config
Config.set("graphics","width","800")
Config.set("graphics","hight","480")
from kivy.core.window import Window
Window.size = (800, 480)

from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.gridlayout import GridLayout
from kivy.uix.button import Button
from kivy.uix.popup import Popup
from kivy.uix.image import Image
from kivy.uix.label import Label
from kivy.uix.screenmanager import ScreenManager, Screen
from kivy.clock import Clock

import mysql.connector
from mysql.connector import Error

import database
from database import checkcnx, alldp, allws, allmfm, registercart, showproduct,
registerbackup

from datetime import date
from datetime import datetime

class MessagePopup(Popup):
    pass

Clock.schedule_interval(lambda x: registerbackup(), 900)

class MainWid(ScreenManager):
    def __init__(self,**kwargs):
        super(MainWid,self).__init__()
        self.cart=['','','','','','','','']
        #afegeixo Widgets principals
        self.DPWid = DPWid(self)
        self.WSWid = WSWid(self)
        self.ScanWid = ScanWid(self)
        self.FMWid = FMWid(self)
        self.ConfirmWid = ConfirmWid(self)
        #creo cada infowid i menuwid per pantalla
        self.InfoWidDP = InfoWid(self)
        self.InfoWidWS = InfoWid(self)
        self.InfoWidScan = InfoWid(self)
        self.InfoWidFM = InfoWid(self)
        self.InfoWidConfirm = InfoWid(self)
        self.InfoWidConfirm.size_hint_y = 0.5
```

```
        self.InfoWidConfirm.size_hint_x = 1
        self.MenuWidDP = MenuWid(self)
        self.MenuWidWS = MenuWid(self)
        self.MenuWidScan = MenuWid(self)
        self.MenuWidFM = MenuWid(self)
        self.MenuWidConfirm = MenuWid(self)
        #self.ScrollFM = ScrollWid(self)
        self.Popup = MessagePopup()
        #self.cart=['dp','ws','sn','lot','mfm','panel', 'date', 'hour']
        #creo pantalles i afegeixo a cada menu el wid principal i la info
        wid = Screen(name='WS')
        wid.add_widget(self.MenuWidWS)
        self.MenuWidWS.add_widget(self.WSWid)
        self.MenuWidWS.add_widget(self.InfoWidWS)
        self.add_widget(wid)
        wid = Screen(name='Scan')
        wid.add_widget(self.MenuWidScan)
        self.MenuWidScan.add_widget(self.ScanWid)
        self.MenuWidScan.add_widget(self.InfoWidScan)
        self.add_widget(wid)
        wid = Screen(name='FM')
        wid.add_widget(self.MenuWidFM)
        #self.ScrollFM.add_widget(self.FMWid)
        self.MenuWidFM.add_widget(self.FMWid)
        self.MenuWidFM.add_widget(self.InfoWidFM)
        self.add_widget(wid)
        wid = Screen(name='DP')
        wid.add_widget(self.MenuWidDP)
        self.MenuWidDP.add_widget(self.DPWid)
        self.MenuWidDP.add_widget(self.InfoWidDP)
        self.add_widget(wid)
        wid = Screen(name='Confirm')
        wid.add_widget(self.ConfirmWid)
        self.add_widget(wid)

        self.goto_dp()


    def goto_dp(self):
        #self.DPWid.CreateDP()
        self.DPWid.list = self.DPWid.createDP()
        #self.DPWid.cols = self.DPWid.col()
        self.current ='DP'

    def goto_ws(self):
        print(self.cart)
        self.InfoWidWS.updateinfo()
        self.WSWid.list = self.WSWid.createWS(self.cart[0])
        self.current = 'WS'
        #self.WSWid.createbtn()

    def goto_scan(self):
        print(self.cart)
        self.InfoWidScan.updateinfo()
        self.current = 'Scan'
```

```
    def goto_fm(self):
        print(self.cart)
        self.InfoWidFM.updateinfo()
        self.FMWid.list = self.FMWid.createFM(self.cart[1])
        self.current = 'FM'


    def goto_confirm(self):
        print(self.cart)
        self.ConfirmWid.updateinfo()
        self.current = 'Confirm'

class ScrollWid(ScrollView):
    def __init__(self, mainwid, **kwargs):
        super(ScrollWid, self).__init__()
        self.mainwid = mainwid
        self.size = self.size

class MenuWid(BoxLayout):
    def __init__(self, mainwid, **kwargs):
        super(MenuWid, self).__init__()
        self.mainwid = mainwid




class InfoWid(BoxLayout):
    def __init__(self,mainwid,**kwargs):
        super(InfoWid,self).__init__()
        self.mainwid = mainwid
        self.orientation = 'vertical'
        self.add_widget(Image(source = 'qiagen_logo.gif', size_hint_y = 0.2))
        self.station = Label(text='Station', size_hint_y=0.075, color=(0, 0, 0,
1), halign = 'left', pos=(0,0))
        self.station.bind(size=self.station.setter('text_size'))
        self.add_widget(self.station)
        self.lot = Label(text='Lot', size_hint_y = 0.075, color = (0, 0, 0, 1),
halign = 'left', pos=(0,0))
        self.lot.bind(size=self.lot.setter('text_size'))
        self.add_widget(self.lot)
        self.cartridge = Label(text='Cartridge nº', size_hint_y = 0.075, color
= (0, 0, 0, 1), halign = 'left', pos=(0,0))
        self.cartridge.bind(size=self.cartridge.setter('text_size'))
        self.add_widget(self.cartridge)
        self.panel = Label(text='Panel', size_hint_y=0.075, color=(0, 0, 0, 1),
halign = 'left', pos=(0,0))
        self.panel.bind(size=self.panel.setter('text_size'))
        self.add_widget(self.panel)
        self.Failure = Label(text='Failure Mode', size_hint_y = 0.075, color =
(0, 0, 0, 1), halign = 'left', pos=(0,0))
        self.Failure.bind(size=self.Failure.setter('text_size'))
        self.add_widget(self.Failure)
        but1 = Button(text='home', size_hint_y = 0.2)
        but1.bind(on_press = lambda x: self.home())
        but2 = Button(text = 'back', size_hint_y = 0.2)
```

```
        but2.bind(on_press = lambda x: self.back())
        self.add_widget(but1)
        self.add_widget(but2)
        self.size_hint_x = 0.25


    def home(self):
        self.mainwid.cart = ['','','','','', '', '', '']
        #borrar widgets
        for elemento in self.mainwid.DPWid.list:
            self.mainwid.DPWid.remove_widget(elemento[1])
        for elemento in self.mainwid.WSWid.list:
            self.mainwid.WSWid.remove_widget(elemento[1])
        for elemento in self.mainwid.FMWid.list:
            self.mainwid.FMWid.remove_widget(elemento[1])
        self.mainwid.goto_dp()
        self.station.text = 'Station ' + str(self.mainwid.cart[1])
        self.lot.text = 'Lot ' + str(self.mainwid.cart[3])
        self.cartridge.text = 'Cartridge nº ' + str(self.mainwid.cart[2][5:9])
        self.panel.text = 'Panel' + str(self.mainwid.cart[5])
        self.Failure.text = 'Failure Mode' + str(self.mainwid.cart[4])

    def back(self):
        if self.mainwid.current == 'DP':
            None
        elif self.mainwid.current == 'WS':
            self.mainwid.cart = ['','','','','', '', '', '']
            for elemento in self.mainwid.DPWid.list:
                self.mainwid.DPWid.remove_widget(elemento[1])
            for elemento in self.mainwid.WSWid.list:
                self.mainwid.WSWid.remove_widget(elemento[1])
            for elemento in self.mainwid.FMWid.list:
                self.mainwid.FMWid.remove_widget(elemento[1])
            self.mainwid.goto_dp()
        elif self.mainwid.current == 'Scan':
            self.mainwid.cart[1] = ''
            for elemento in self.mainwid.WSWid.list:
                self.mainwid.WSWid.remove_widget(elemento[1])
            for elemento in self.mainwid.DPWid.list:
                self.mainwid.DPWid.remove_widget(elemento[1])
            for elemento in self.mainwid.FMWid.list:
                self.mainwid.FMWid.remove_widget(elemento[1])
            self.mainwid.goto_ws()
        elif self.mainwid.current == 'FM':
            self.mainwid.cart[2] = ''
            self.mainwid.cart[3] = ''
            self.mainwid.goto_scan()
        self.station.text = 'Station ' + str(self.mainwid.cart[1])
        self.lot.text = 'Lot ' + str(self.mainwid.cart[3])
        self.cartridge.text = 'Cartridge nº ' + str(self.mainwid.cart[2][5:9])
        self.panel.text = 'Panel' + str(self.mainwid.cart[5])
        self.Failure.text = 'Failure Mode' + str(self.mainwid.cart[4])

    def updateinfo(self):
        self.station.text = 'Station ' + "\n" + str(self.mainwid.cart[1])
        self.lot.text = 'Lot ' + "\n" + str(self.mainwid.cart[3])
```

```
        self.cartridge.text = 'Cartridge nº ' + "\n" +
str(self.mainwid.cart[2][5:9])
        self.panel.text = 'Panel' + "\n" + str(self.mainwid.cart[5])
        self.Failure.text = 'Failure Mode' + "\n" + str(self.mainwid.cart[4])

class DPWid(GridLayout):
    def __init__(self, mainwid, **kwargs):
        super(DPWid,self).__init__()
        self.mainwid = mainwid
        self.padding = [30,30,30,30]
        self.spacing = [30,30]
        self.cols = 3
        self.rows = 3

    def createDP(self):
        lista = []
        for dp in alldp():
            but = Button(text = alldp()[dp], size_hint_x=None,
size_hint_y=None, width=175, height=125)
            but.bind(on_press=lambda x: self.selDP())
            but.id = dp
            lista.append([dp,but])
            self.add_widget(but)
            #dp es DP01: WS de XX a YY
            #print(alldp()[dp])
        return lista

    def selDP(self):
        dp = ''
        for element in self.list:
            if element[1].last_touch != None:
                dp = element[0]

        self.mainwid.cart[0] = dp
        self.mainwid.goto_ws()
        for elemento in self.list:
            self.remove_widget(elemento[1])



class WSWid(GridLayout):
    def __init__(self, mainwid,**kwargs):
        super(WSWid , self).__init__()
        self.__name__= 'WS'
        self.mainwid = mainwid
        #self.size = [800, 480]
        self.padding = [30, 30, 30, 30]
        self.spacing = [30, 30]
        self.list = []
        self.rows = 3
        self.orientation = 'vertical'


    def createWS(self,dp):
        lista = []
        for ws in allws(dp):
```

```
            but = Button(text=ws+':'+allws(dp)[ws], size_hint_y=None,
width=500, height=100)
            but.id = ws
            but.bind(on_press=lambda x: self.selWS())
            lista.append([ws, but])
            self.add_widget(but)
            # ws es WS01: description
            # print(alldp()[dp])
        #print(lista)
        return lista

    def selWS(self):
        ws=''
        for element in self.list:
            if element[1].last_touch != None:
                ws = element[0]

        self.mainwid.cart[1] = ws
        self.mainwid.goto_scan()
        for elemento in self.list:
            self.remove_widget(elemento[1])


class ScanWid(GridLayout):
    def __init__(self,mainwid,**kwargs):
        super(ScanWid,self).__init__()
        self.mainwid = mainwid
        #self.size = [800, 480]
        self.padding = [30, 30, 30, 30]
        self.spacing = [30, 30]


    def scan(self, input):
        print('Cartridge Scanned')
        # no tener escuchando todo el rato poner clock
        info = input
        sn = ''
        product = ''
        if len(input) == 23:
            gtin = info[:14]
            product = showproduct(gtin)
            sn = info[14:]

        elif len(input) == 269:
            gtin = info[2:16]
            product = showproduct(gtin)
            sn = info[34:43]

        year = str(date.today().year)
        day = str(datetime.now())[:10]
        hour = str(datetime.now())[11:19]
        # input example SN  040532280339054922513 91
        #                   040532280339054922513 98
        # input example DataMatrix
0104053228033905101902251720052721492251391200191999999999999999999991100921075
9999999999999999911009310759999999999999999911009410759999999999999999911009510759
```

```
99999999999999911009610759999999999999991100971075999999999999999911009899999999999999999999111025001250

        lot = year[2:4]+'0'+sn[2:5]
        if len(input) != 23 and len(input) != 269:
            lot = ''
            content = Button(text ='Any Cartridge registred, Scan
again',size=(400,400))
            content.bind(on_press = lambda x: self.mainwid.goto_scan())
            popup = Popup(title='Error',content=content, auto_dismiss=True,
size=(400,400))
            content.bind(on_press = popup.dismiss)
            popup.open()
        self.mainwid.cart[2] = sn
        self.mainwid.cart[3] = lot
        self.mainwid.cart[5] = product
        self.mainwid.cart[6] = day
        self.mainwid.cart[7] = hour
        #print(self.mainwid.cart)

        self.mainwid.goto_fm()

class FMWid(GridLayout):
    def __init__(self,mainwid,**kwargs):
        super(FMWid,self).__init__()
        self.mainwid = mainwid
        #self.size = [800, 480]
        self.list = []
        self.padding = [30, 30, 30, 30]
        #self.spacing = [30, 30]
        self.cols = 2



    def createFM(self, ws):
        lista = []
        texto = ""
        for mfm in allmfm(ws):
            texto = mfm+":"+"\n" + allmfm(ws)[mfm]
            but = Button(text=texto)
            but.halign = 'center'
            but.id = mfm
            but.bind(on_press=lambda x: self.selFM())
            lista.append([mfm, but])
            self.add_widget(but)
            # ws es WS01: description
        return lista

    def selFM(self):
        # print(self.mainwid.cart)
        # self.mainwid.cart[1] = ws
        mfm=''
        for element in self.list:
            if element[1].last_touch != None:
                mfm = element[0]
```

```
        self.mainwid.cart[4] = mfm
        self.mainwid.goto_confirm()
        for elemento in self.list:
            self.remove_widget(elemento[1])


class ConfirmWid(BoxLayout):
    def __init__(self,mainwid,**kwargs):
        super(ConfirmWid,self).__init__()
        self.mainwid = mainwid
        #self.size = [800, 480]
        self.orientation = 'vertical'
        self.add_widget(Image(source='qiagen_logo.gif', size_hint_y=0.25))
        self.station = Label(text='Station', size_hint_y=0.075, color=(0, 0, 0,
1), halign='center', pos=(0, 0))
        self.station.bind(size=self.station.setter('text_size'))
        self.add_widget(self.station)
        self.lot = Label(text='Lot', size_hint_y=0.075, color=(0, 0, 0, 1),
halign='center', pos=(0, 0))
        self.lot.bind(size=self.lot.setter('text_size'))
        self.add_widget(self.lot)
        self.cartridge = Label(text='Cartridge nº', size_hint_y=0.075,
color=(0, 0, 0, 1), halign='center', pos=(0, 0))
        self.cartridge.bind(size=self.cartridge.setter('text_size'))
        self.add_widget(self.cartridge)
        self.panel = Label(text='Panel', size_hint_y=0.075, color=(0, 0, 0, 1),
halign='center', pos=(0, 0))
        self.panel.bind(size=self.panel.setter('text_size'))
        self.add_widget(self.panel)
        self.Failure = Label(text='Failure Mode', size_hint_y=0.075, color=(0,
0, 0, 1), halign='center', pos=(0, 0))
        self.Failure.bind(size=self.Failure.setter('text_size'))
        self.add_widget(self.Failure)
        self.buts = BoxLayout(size_hint_y=0.25, padding = 15, spacing = 15)
        self.buts.orientation = 'horizontal'
        but1 = Button(text='Confirm', background_color = [1,0,0,0,5])
        but1.bind(on_press=lambda x: self.confirm())
        but2 = Button(text='Reject', background_color = [0,1,0,0,5])
        but2.bind(on_press=lambda x: self.reject())
        self.buts.add_widget(but1)
        self.buts.add_widget(but2)
        self.add_widget(self.buts)


    def updateinfo(self):
        self.station.text = 'Station ' + "\n" + str(self.mainwid.cart[1])
        self.lot.text = 'Lot ' + "\n" + str(self.mainwid.cart[3])
        self.cartridge.text = 'Cartridge nº ' + "\n" +
str(self.mainwid.cart[2][5:9])
        self.panel.text = 'Panel' + "\n" + str(self.mainwid.cart[5])
        self.Failure.text = 'Failure Mode' + "\n" + str(self.mainwid.cart[4])

    def confirm(self):
        registercart(self.mainwid.cart)
        self.mainwid.cart = ['', '', '', '', '', '', '', '']
        content = Button(text='Done, the Cartridge has been successfully
```

```
registered', size=(400, 400))
        popup = Popup(title='Done', content=content, auto_dismiss=True,
size=(400, 400))
        content.bind(on_press=popup.dismiss)
        popup.open()
        self.mainwid.goto_dp()

    def reject(self):
        self.mainwid.cart = ['', '', '', '', '', '', '', '']
        self.mainwid.goto_dp()




class MainApp(App):
    def build(self):
        return MainWid()

if __name__ == '__main__':
    MainApp().run()
```

11.2. Codi database.py

```
config = {
  'user': 'root',
  'password': 'Welcome13',
  'host': 'localhost',
  'database': 'mfm',
  'raise_on_warnings': True
}
def checkcnx():
    try:
        cnx = mysql.connector.connect(**config)

    except mysql.connector.Error as err:
        if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
            print("Something is wrong with your user name or password")
            return False
        elif err.errno == errorcode.ER_BAD_DB_ERROR:
            print("Database does not exist")
            return False
        else:
            print("error")
            return False
    else:
        cnx.close()
        return True

def alldp():
    dp={}
    if checkcnx():
        cnx = mysql.connector.connect(**config)
        cursor = cnx.cursor()
        cursor.execute("SELECT * FROM discardstation")
```

```
        dstations=cursor.fetchall()
        for dstation in dstations:
            dp[str(dstation[1])]=dstation[3]
        return dp

def allws(ds):
    ws={}
    if checkcnx():
        cnx = mysql.connector.connect(**config)
        cursor = cnx.cursor()
        cursor.execute("SELECT * FROM workstation")
        wstations=cursor.fetchall()
        for wstation in wstations:
            if wstation[4]==ds:
                ws[str(wstation[1])]=wstation[2]
        return ws

def allmfm(ws):
    #print(ws)
    n=int(ws[2:4])
    if 0 < n < 3:
        ws = 'WS02'
    elif 3 <= n < 8:
        ws = 'WS07'
    elif n == 8:
        ws = 'WS08'
    elif n == 9:
        ws = 'WS09'
    elif 10 <= n < 14:
        ws = 'WS13'
    elif 14 <= n < 17:
        ws = 'WS16'
    elif n == 17:
        ws = 'WS17'
    elif n == 18:
        ws = 'WS18'

    mfm={}
    if checkcnx():
        cnx = mysql.connector.connect(**config)
        cursor = cnx.cursor()
        cursor.execute("SELECT * FROM mfm")
        mfmodes = cursor.fetchall()
        for mfmode in mfmodes:
            if mfmode[4] == ws:
                mfm[str(mfmode[1])] = mfmode[2]
        return mfm


def showmfm(a):
    b=''
    if 0 < a <10 and a in (2,7,9,13,16,17,18):
        b='0'+str(a)
    if 10 <= a < 19 and a in (2,7,9,13,16,17,18):
        b=str(a)
    else:
```

```
        print('WS do not exists')
    ws='WS'+b
    if checkcnx():
        cnx = mysql.connector.connect(**config)
        cursor = cnx.cursor()
        cursor.execute("SELECT * FROM mfm")
        mfm=cursor.fetchall()
        for mf in mfm:
            if ws == mf[4]:
                print(mf[2])
    cnx.close()

def checkcart(cart):
    dp = cart[0]
    ws = cart[1]
    sn = cart[2].strip(' ')
    lot = cart[3]
    mfm = cart[4]
    panel = cart[5]
    date = cart[6]
    time = cart[7]
    if checkcnx():
        cnx = mysql.connector.connect(**config)
        cursor = cnx.cursor()
        cursor.execute("SELECT * FROM manufacturing")
        cartridges = cursor.fetchall()
        for cartr in cartridges:
            if str(sn) == str(cartr[1]) and date == cartr[13] and time ==
cartr[14]:
                return True

        return False


def registercart(cart):
    dp = cart[0]
    ws = cart[1]
    sn = cart[2]
    lot = cart[3]
    mfm = cart[4]
    panel = cart[5]
    date = cart[6]
    time = cart[7]
    if not checkcart(cart):
        backup = open("backup.txt", "a")
        backup.write(str(dp)+' '+str(ws)+' '+str(sn)+' '+str(lot)+'
'+str(mfm)+' '+str(panel)+' '+str(date)+' '+str(time)+'\n')
        backup.close()
    if checkcnx() and not checkcart(cart):
        cnx = mysql.connector.connect(**config)
        cursor = cnx.cursor()
        mySql_insert_query = """INSERT INTO manufacturing (dp, ws, sn, lot,
mfm, panel, date, time)
                                       VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
"""
        records_to_insert = (dp, ws, sn, lot, mfm, panel, date, time)
```

```
        cursor.execute(mySql_insert_query, records_to_insert)
        cnx.commit()
    elif checkcnx() and checkcart(cart):
        print('Cartridge already scanned')

def registerbackup():
    backup = open("backup.txt", "r")
    cartridges = backup.readlines()
    for cartridge in cartridges:
        cart = cartridge.rsplit(' ')
        dp = cart[0]
        ws = cart[1]
        sn = cart[2]
        lot = cart[3]
        mfm = cart[4]
        panel = cart[5]+' '+cart[6]+' '+cart[7]
        date = cart[8]
        time = cart[9]
        if checkcnx() and not checkcart(cart):
            cnx = mysql.connector.connect(**config)
            cursor = cnx.cursor()
            mySql_insert_query = """INSERT INTO manufacturing (dp, ws, sn, lot,
mfm, panel, date, time)
                                    VALUES (%s, %s, %s, %s, %s, %s, %s,
%s) """
            records_to_insert = (dp, ws, sn, lot, mfm, panel, date, time)

            cursor.execute(mySql_insert_query, records_to_insert)
            cnx.commit()

def showproduct(gtin):
    if checkcnx():
        cnx = mysql.connector.connect(**config)
        cursor = cnx.cursor()
        cursor.execute("SELECT * FROM products")
        products=cursor.fetchall()
        for product in products:
            if gtin == product[1]:
                return product[3]
```

11.3. Codi main.kv

```
<MainWid>:

    canvas:
        Color:
            rgb: 1,1,1
        Rectangle:
            pos: self.pos
            size: 800,480

<MenuWid>:
    canvas:
        Color:
```

```
            rgb: 1,1,1
        Rectangle:
            pos: self.pos
            size: self.size



#ara el menu es queda sueprposat, no queden posats en el mateix pla, important!

<DPWid>:

<WSWid>:

<ScanWid>:
    name: 'Scan'

    cols:1
    rows:3
    spacing: [30, 30]
    padding: 30
    size: 720,480
    pos: self.pos
    canvas:
        Color:
            rgb: 1,1,1
        Rectangle:
            size:self.size
            pos:self.pos

    Label:
        text: "Scrap Register"
        color: 0,0,0,1
        font_size: 40
        bold: True


    Image:
        source: 'qiagen_logo.gif'
        pos: self.pos
        size: self.size

    TextInput:
        id: input
        hint_text: 'SCAN'
        font_size: (root.width**2 + root.height**2) / 12**4
        multiline: False
        on_focus: self.text = ''
        on_text_validate: root.scan(input.text)

    #Button:
        #font_size: (root.width**2 + root.height**2) / 12**4
        #text: "SCAN"
        #on_press: root.scan(input.text)
```